



Europäisches Patentamt
European Patent Office
Office européen des brevets



Publication number: **0 632 377 A1**

EUROPEAN PATENT APPLICATION

Application number: **94110016.6**

Int. Cl.⁸: **G06F 11/00**

Date of filing: **28.06.94**

Priority: **30.06.93 US 85621**

Date of publication of application:
04.01.95 Bulletin 95/01

Designated Contracting States:
DE FR GB

Applicant: **MICROSOFT CORPORATION**
One Microsoft Way
Redmond,
Washington 98052-6399 (US)

Inventor: **Blake, Russel P.**

5623-294th Avenue N.E.
Carnation,
Washington 98012 (US)
Inventor: **Day, Robert F.**
19906 Filbert Drive
Bothell,
Washington 98012 (US)

Representative: **Patentanwälte Grünecker,**
Kinkeldey, Stockmair & Partner
Maximilianstrasse 58
D-80538 München (DE)

Method for testing a message-driven operating system.

A simulation system to simulate the execution of a computer program. The computer program is developed for invoking operating system functions of a first operating system. Each operating system function performs a behavior in accordance with passed parameters. The simulation system generates a log during the execution of the computer program under control of the first operating system. The log includes an indication of each invocation of an operating system function by the computer program and an indication of each parameter passed to the operating system function by the computer program and the current time. The logged execution is then simulated by the simulation system on a second operating system. The simulation system invokes an operating system function of the second operating system to perform a behavior similar to the behavior performed by each logged invocation of the operating system function of the first operating system in accordance with the passed parameters. Comparison of the functionality, reliability, and performance of the two systems are thereby enabled.

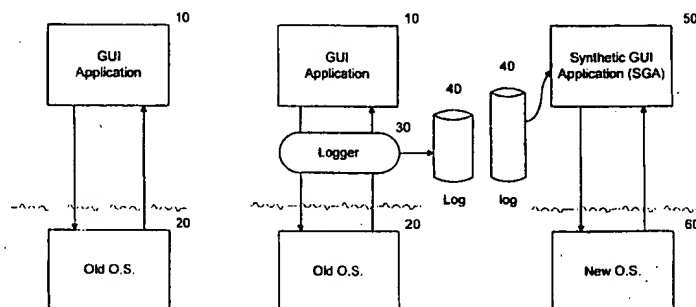


FIG. 2A

FIG. 2B

FIG. 2C

EP 0 632 377 A1

Technical Field

This invention relates generally to a computer method and system for simulating the execution of a computer program.

5

Background of the Invention

Computer operating systems are very complex computer programs. When developing or modifying an operating system, it is critical that the operating system be thoroughly tested. The testing of an operating system typically involves several testing phases. First, the programmer who writes a program for the operating system performs "unit testing." This unit testing ensures that the program functions as intended by the programmer. Second, the programmers who developed various programs perform integration testing. This integration testing ensures that the various programs function together correctly. Third, the developer of the operating system performs alpha testing of the operating system. During alpha testing, application programs are executed with the operating system and any anomalies are logged for later correction. Finally, end users of the operating system perform beta testing. The beta testing ensures that the operating system will function correctly in the end user's environment.

The testing of an operating system can be very time consuming and expensive. Furthermore, it is virtually impossible to ensure that the operating system is error free. Generally, operating system developers concentrate on ensuring that the operating system will function correctly with "standard applications." A standard application is an application program that a typical user of the operating system may use. By testing with these standard applications, a developer can help ensure the operating system will function correctly in most typical situations.

Certain operating systems are referred to as message-driven operating systems. One such operating system is Windows 3.1, developed by Microsoft Corporation. A description of Windows 3.1 is provided in the Software Development Kit for Windows 3.1, which is available from Microsoft Corporation and is hereby incorporated by reference. The Windows operating system provides a windowing environment for applications that support a graphical user interface (GUI). Figure 1 is a block diagram illustrating the messaging architecture of a typical message-driven operating system. An application program 110 contains a main procedure 111 and a window procedure 112. When the application program 110 is executed under the control of the operating system, control is passed to the main procedure 111. The main procedure 111 typically creates and displays a window and then enters a message loop 113. When executing the message loop, the application program 110 waits to receive a message from the operating system 120 indicating an external event (e.g., key down). The messages received by the message loop are referred to as posted messages. When a message is received, the application program 110 processes the message by requesting the operating system 120 to dispatch the message to the appropriate window procedure. The application program 110 includes a window procedure 112 for each window that is displayed on display monitor 130. A window procedure is invoked by the operating system when a message is dispatched to that window or when the operating system sends (discussed below) a message to that window. The window procedure decodes the message and processes the message accordingly. For example, a message dispatched to the window procedure may indicate that a character has been depressed on the keyboard when the window has the focus. A window that has the focus receives all keyboard and mouse inputs.

The operating system 120 provides various functions to application programs that provide services to the application programs. These functions may include: RegisterClass, CreateWindow, ShowWindow, GetMessage, DispatchMessage, and DefWindowProc. These functions are collectively referred to as the application programming interface (API) provided by the operating system, and each function may be individually referred to as an API. During execution of the application program 110, the application program invokes the various functions provided by the operating system. These functions are typically stored in a dynamic link library. When the application program is initially loaded into memory, it dynamically links to each of the functions it uses. As shown by the main procedure 111, the application program 110 initially invokes the function RegisterClass to register a window class with the operating system. Each window class has an associated window procedure for processing messages that are sent to a window. The operating system maintains a window class table 122, which correlates a window class with its window procedure. When a window class is registered, the operating system stores the address of the window procedure in the window class table 122. When a message is to be sent to a window, the operating system invokes the associated window procedure passing it various parameters including the type of the message.

A window procedure is a type of a callback routine. A callback routine is a routine that is part of the application program but is invoked directly by the operating system. The application program provides the

operating system with the address of a callback routine that is developed to perform application-specific processing. The operating system then invokes the callback routine to perform the processing.

The operating system also maintains a message queue 121 for the application program 110. The message queue 121 contains messages that are posted to the application program. Each invocation of the function GetMessage in the message loop 113 retrieves a message from the message queue 121. The posted messages in the message queue typically correspond to external events such as mouse movement. When the operating system detects mouse movement over the window on the display monitor 130, the operating system posts a message to the message queue for the application program. The application program during the message loop retrieves each posted message and invokes the function DispatchMessage to process the message. The function DispatchMessage determines which window procedure the message is directed to and sends the message to that window by invoking its window procedure. Not all messages are posted, dispatched, and then sent to the window procedure. The operating system sometimes sends messages directly to a window procedure, without first posting the message to the message queue. For example, when a window is first created, the operating system may send a create message (WM_CREATE) to the window procedure for that window. This message allows the window procedure to perform initialization of the window.

Summary of the Invention

It is an object of the present invention to provide a method and system for simulating the execution of a computer program to facilitate the testing of an operating system.

It is another object of the present invention to provide a method and system for testing an operating system using application programs that have not yet been converted to execute under the operating system.

These and other objects, which will become apparent as the invention is more fully described below, are provided by a method and system for simulating the execution of a computer program. In a preferred embodiment, a computer system simulates an execution of a client program that requests services of a first server program. During execution of the client program, the requests for services are logged. A simulation program receives the logged requests for services and requests a second server program to simulate the behavior of the requested service. In another preferred embodiment, a simulation system simulates the execution of a computer program. The computer program is developed for invoking operating system functions of a prior operating system. Each operating system function performs a behavior in accordance with passed parameters. The simulation system generates a log during the execution of the computer program under control of the prior operating system. The log includes an indication of each invocation of an operating system function by the computer program and an indication of each parameter passed to the operating system function by the computer program. The logged execution is then simulated on a new operating system. The simulation system invokes an operating system function of the new operating system to perform a behavior similar to the behavior performed by the logged invocation of the operating system function of the prior operating system in accordance with the passed parameters.

Brief Description of the Drawings

Figure 1 is a block diagram illustrating the messaging architecture of a typical message-driven operating system.

Figures 2A, 2B, and 2C are block diagrams illustrating the recording of the interaction and the simulation of an application program.

Figure 3 is a block diagram illustrating a preferred architecture of the logger.

Figure 4 is a flow diagram of a sample substitute function.

Figure 5 is a flow diagram of a routine to log a function invocation.

Figure 6 is a flow diagram illustrating the recording of a typical integer parameter that is passed by value.

Figure 7 is a flow diagram illustrating a routine that records a parameter that is passed as a pointer to a buffer.

Figure 8 is a flow diagram of the processing of a parameter that points to a callback routine.

Figure 9 is a flow diagram of a typical substitute callback routine.

Figure 10A is an overview diagram illustrating the synthetic GUI application.

Figure 10B is a flow diagram of the synthetic GUI application program.

Figure 11 is a flow diagram of the routine GenerateSendMessageFile.

Figure 12 is a flow diagram of the routine GeneratePostedMessageFile.

Figure 13 is a flow diagram of the SGAEngine routine.

Figure 14 is a flow diagram of the routine FindMessageInSentMessageFile.

Figure 15 is a flow diagram of the routine SimulatePostedMessage.

5 Figure 16 is a flow diagram of a routine that simulates the posting of the WM_KEYDOWN message.

Figure 17 is a flow diagram of a routine that simulates a WM_MOUSEMOVE message.

Figure 18 is a flow diagram of a thunk window procedure.

Figure 19 is a flow diagram of the routine ThunkRegisterClass.

Figure 20 is a flow diagram of the routine ThunkCreateWindow.

10 Figure 21 is a flow diagram of the procedure ThunkDestroyWindow.

Figure 22 is a flow diagram of a template thunk function.

Figure 23 is a block diagram illustrating an alternate embodiment of the present invention.

Detailed Description of the Invention

15

The present invention provides a method and system for simulating the execution of an application program. In a preferred embodiment, the simulation system first records the interaction between the application program and an existing operating system (an old operating system) during an execution of the application program. The simulation system then takes this recorded interaction and simulates the

20

interaction with a new operating system. To record the interaction, the simulation system executes the application program under the control of the old operating system. During execution of the application program, a logger portion of the simulation system records each invocation of an operating system function by the application program and records each invocation of a callback routine by the operating system. The logger also records all the parameters

25

passed to each function and all the parameters returned by each callback routine. The logger also preferably records all parameters returned by each function and all parameters passed to each callback routine. The logger records this information in a log file.

30

The simulation system then simulates the execution of the application program on a new operating system. A synthetic GUI application (SGA) of the simulation system inputs the log file and invokes the functions of the new operating system to effect the behavior of the logged function invocations. The SGA also provides thunk callback routines for the new operating system to invoke. The thunk callback routines simulate the behavior of the real callback routines of the application program. The SGA also simulates the occurrence of the real events (e.g., key down) that occurred during the execution of the application program. By effecting a behavior in the new operating system that corresponds to the behavior that

35

occurred during the execution of the application program under the old operating system, the execution of the application program on the new operating system is simulated.

40

Testers of the new operating system can then compare the output of the simulation to ensure that it correctly corresponds to the execution of the application program under the old operating system. Also, the return parameters from the functions of the new operating system can be compared to the recorded returned parameters from the function of the old operating system to determine whether the new operating system is working correctly. Also, the time taken to perform the functions on the new operating system can be compared to the time taken to perform the functions in the old operating system.

45

Figures 2A, 2B, and 2C are block diagrams illustrating the recording of the interaction and the simulation of an application program. Figure 2A illustrates the interaction between application program 210 and an old operating system 220. The GUI application program 210 invokes the functions of the old operating system 220, and the old operating system 220 invokes the callback routines of the GUI application program 210. Figure 2B illustrates the logging of the invocations of the functions and the callback routines during the execution of the application program 210. When the application program 210 invokes a function, the logger 230 records the passed parameters for the function and the function name to a log file 240. When the function returns to the application program, the logger also records all returned parameters to the log file. When the old operating system 220 invokes a callback routine, the logger 230 records a callback routine identifier and the passed parameters. When the callback routine returns to the old operating system 220, the logger records the returned parameters to the log file. The time associated with each interaction is recorded. Figure 2C is a block diagram illustrating the simulation of the execution of the

50

application program 210 that is recorded in the log file. To simulate the execution of the application program 210, the synthetic GUI application 250 (SGA) inputs the log file that records the execution. The SGA simulates each function invocation represented in the log file by invoking none, one, or more functions of the new operating system 260. The SGA simulates the behavior of the function invocation based on the

55

recorded passed parameters. The SGA 250 also provides a thunk callback routine for each real callback routine of the application program. The SGA 250 passes these thunk callback routines to the new operating system 260. When the new operating system 260 invokes a thunk callback routine, the thunk callback routine simulates the processing of the real callback routine based on the information in the log file.

5 Specifically, the thunk callback routine may simulate the posting and sending of messages and the invoking of functions that occurred during the execution of the real callback routine.

The Logger

10 Figure 3 is a block diagram illustrating a preferred architecture of the logger. For each function 221 that the old operating system 220 provides, the logger provides a substitute function 311. The logger also provides a substitute callback routine 312 for each callback routine of the application program. When the GUI application program 210 is initially loaded, all function calls by the application program link to a substitute function 311, rather than the real function 221 in the old operating system. When the application

15 program 210 invokes a function, the substitute function records the invocation to a log file along with each passed parameter and the current time. The substitute function then invokes the real function of the old operating system 220 with the passed parameters. When the real function returns to the substitute function, the substitute function records the return along with the returned parameters to the log file and the current time. The substitute function then returns to the application program with the parameters returned by the

20 real function. In this way, the recording of the log file is functionally transparent to the application program.

The real functions of the old operating system are typically stored in a dynamic link library. The executable file containing the application program contains the name of each dynamic link library that the application program invokes. During initial loading of the application program, each invocation of a real function is bound to the real function in the dynamic link library. In a preferred embodiment, each dynamic

25 link library with real functions is associated with a substitute dynamic link library and is given a name with the same number of characters as the name of the "real" dynamic link library and a slight variation of the name of the real dynamic link library. For example, the name "Zernel" may be given to the substitute dynamic link library corresponding to the real dynamic link library "Kernel." When the execution of the application program is to be logged, each real dynamic link library name in the executable file is replaced

30 by a substitute dynamic link library name. Thus, when the application program is loaded, the application program dynamically links to the substitute dynamic link libraries, rather than the real dynamic link libraries.

The addresses of callback routines are typically specified to the old operating system by a parameter passed to a function. For example, the address of a window procedure callback routine is specified to the old operating system as a parameter when invoking the function RegisterClass. The substitute function

35 associated with each real function, that is passed as a callback routine, substitutes a substitute callback routine for the real callback routine. The substitute function invokes the real function specifying the substitute callback routine 312 rather than the real callback routine 211. When the old operating system 220 sends a message to the application program 210, it invokes the substitute callback routine 312. The substitute callback routine records the invocation of the callback routine along with the passed parameters

40 and the current time. The substitute callback routine then invokes the real callback routine 211 with the passed parameters. When the real callback routine returns to the substitute callback routine, the substitute callback routine records the return along with the returned parameters. The substitute callback routine then returns to the old operating system with the returned parameters.

When a substitute function or a substitute callback routine records its invocation, it also records a

45 nesting level. The nesting level indicates the level of invocations of the functions and callback routines. For example, when an application program invokes the function DispatchMessage, the nesting level is 1 because no other function or callback routine is currently invoked. If, during the execution of the function DispatchMessage, the function invokes a callback routine, then the nesting level of the callback routine is 2. If that callback routine then invokes a function (e.g., function DefWindowProc), then the invocation of that

50 function is at nesting level 3. If the function at nesting level 3 invokes a callback routine, then the nesting level of the callback routine is 4, and so on. Alternatively, the nesting level of an invocation can be determined from the log after completion of the execution of the application program, rather than during execution.

Figure 4 is a flow diagram of a sample substitute function. The substitute function has the same calling

55 prototype (that is, the same function type and the same number and type of parameters) as the real function. In a preferred embodiment, the substitute function is generated automatically from a "header" file that defines the original function to the application. The substitute function records the invocation of the function along with the passed parameters and the current time and the return of the function along with the

returned parameters and the current time. The substitute function invokes the real function. The substitute function ensures that the real function is passed the same parameters that it receives, and ensures that the application program is returned the same parameters that the real function returns. In step 401, the substitute function saves the current state of the CPU, which may include saving the registers. The substitute function ensures that CPU state is restored to this saved state before the real function is invoked. In step 402, the substitute function increments the nesting level. In step 402A, the substitute function retrieves the current time. In step 403, the substitute function records the function invocation and passed parameters and the current time. In step 404, the substitute function restores the state of the CPU to the state saved in step 401. In step 405, the substitute function invokes the real function with the passed parameters. In step 406, the substitute function saves the state of the CPU upon return from the real function. The substitute function ensures that the CPU state is restored to this saved state before the substitute function returns. In step 406A, the substitute function retrieves the current time. In step 407, the substitute function records the return and returned parameters and the current time. In step 408, the substitute function decrements the nesting level. In step 409, the substitute function restores the CPU state saved in step 406 and returns to the application program.

Figure 5 is a flow diagram of a routine to record a function invocation to a log file. The routine writes an identification of the real function (e.g., function name) to the log file along with all the parameters passed to the real function and the current time. The log file preferably includes an entry (e.g., a line) for each recorded invocation and return of a function or callback routine. Each entry identifies whether it corresponds to an invocation or return of a function or callback routine. In step 501, the routine writes the nesting level and function invocation identifier (e.g., "APICALL") to the log file and the current time. In step 502, the routine writes the function name (e.g., "RegisterClass") to the log file. In steps 503 through 505, the routine loops writing the passed parameters to the log file. The routine must write all the information to the log file that the real function may use when it executes. For example, if the real function prints a buffer of data to the display monitor, then the buffer may be pointed to by a pointer. The routine copies all the data in the buffer to the log file, not just the pointer. This actual data can then be redisplayed during simulation. In step 503, if all the parameters have been processed, then the routine returns, else the routine continues at step 504. In step 504, the routine retrieves the next parameter. In step 505, the routine processes the retrieved parameter and writes the data associated with the parameter to the log file, and then loops to step 503.

Figure 6 is a flow diagram illustrating the recording of a typical integer parameter that is passed by value. The value of the parameter is retrieved and written to the log file.

Figure 7 is a flow diagram illustrating a routine that records a parameter that is passed as a pointer to a buffer. The routine not only records the data pointed to by the passed pointer (the buffer), but also records any data pointed to by pointers in the buffer. In step 701, the routine writes a left brace to the log file. The left brace indicates the start of data that is pointed to. In step 702, if the next data item is a pointer, then the routine continues at step 704, else the routine continues at step 703. In step 703, the routine writes the non-pointer parameter to the log file. In step 704, the routine recursively calls itself to process the pointer. In step 705, if all the parameters have been processed, then the routine continues at step 706, else the routine loops to step 702. In step 706, the routine writes a right brace to the log file and returns.

Figure 8 is a flow diagram of the processing of a parameter that points to a callback routine. In a preferred embodiment, each substitute function that is passed a callback routine includes an array of substitute callback routines. Each time the substitute function is invoked, it selects the next substitute callback routine in the array as the substitute for the passed callback routine. The substitute function also maintains a table that correlates each real callback routine to its substitute callback routine. For example, the substitute function for the function RegisterClass preferably contains an array of identical substitute window procedures. When each new window class is registered, the next unused substitute window procedure in the array is selected. Alternatively, a substitute function could create the substitute callback routines as needed. In step 801, the routine selects an unused substitute callback routine. In step 802, the routine maps the real callback routine to the substitute callback routine. In step 803, the routine substitutes the address of the substitute callback routine for the address of the real callback routine in the parameter list and returns.

Figure 9 is a flow diagram of a typical substitute callback routine. The typical substitute callback routine is analogous to the typical substitute function as shown in Figure 4. In step 901, the substitute callback routine saves the CPU state. In step 902, the substitute callback routine increments the nesting level. In step 902A, the substitute callback routine retrieves the current time. In step 903, the substitute callback routine records the invocation of the callback routine along with the passed parameters and current time to the log file. In step 904, the substitute callback routine finds the address of the associated real callback routine. In step 905, the substitute callback routine restores the CPU state to the state that was saved in

step 901. In step 906, the substitute callback routine invokes the real callback routine. In step 907, the substitute callback routine saves the CPU stat returned by the real callback routine. In step 907A, the substitute callback routine retrieves the current time. In step 908, the substitute callback routine records the return of the callback routine along with the returned parameters and current time to the log file. In step 909, the substitute callback routine decrements the nesting level. In step 910, the substitute callback routine restores the CPU state to the state that was saved in step 907 and the substitute callback routine returns to the invocation by the operating system.

Table 1 contains a sample application program written in the "C" programming language. The application program creates and displays an instance of a window class named "Generic2Class" with the window procedure named "MainWndProc". The window procedure handles the messages WM_COMMAND, WM_DESTROY, WM_PAINT, AND WM_LBUTTONDOWN. All other messages are passed by the window procedure to the operating system using the function DefWindowProc. The window procedure when sent a WM_COMMAND message with the IBM_ABOUT parameter invokes the function DialogBox with a pointer to the callback routine to handle messages to the dialog box.

Table 2 contains a section of a log file for an execution of the sample application program of Table 1. Table 2 contains an entry (line) for each function and callback routine invocation and return. Each entry contains a line number (for reference), a nesting level, a vertical bar, timing information, another vertical bar, an identifier of the type of entry, the function name or callback routine identifier, and the parameters. For example, line number 7 contains an entry corresponding to an invocation of the function CreateWindow. The nesting level is 1. The function was invoked at time "942102B3". The entry type is "APICALL", which indicates the invocation of a function. The parameters are "Generic2Class", "Generic Sample Application", "CF00", etc. The entry corresponding to the return of the function CreateWindow is at line 24. Note that each nesting level in between lines 7 and 24 is at a level greater than 1. During the invocation of the function CreateWindow, the operating system sent the message "WM_GETMINMAXINFO" to the callback routine passed to the function CreateWindow as indicated at line 8. An entry type corresponding to the invocation of a callback routine is "MSGCALL".

Table 1

30

/*.....

PROGRAM: Generic.c

35

PURPOSE: Generic template for Windows applications

FUNCTIONS:

40

WinMain() - calls initialization function, processes message loop

InitApplication() - initializes window data and registers window

InitInstance() - saves instance handle and creates main window

MainWndProc() - processes messages

About() - processes messages for "About" dialog box

45

50

55

COMMENTS:

Windows can have several copies of your application running at the same time. The variable `hInst` keeps track of which instance this application is so that processing will be to the correct window.

```
...../
HANDLE hInst;           /* current instance */
HWND hWndSave;
```

```
...../
FUNCTION: WinMain(HANDLE, HANDLE, LPSTR, int)
```

PURPOSE: call initialization function, processes message loop

COMMENTS:

Windows recognizes this function by name as the initial entry point for the program. This function calls the application initialization routine, if no other instance of the program is running, and always calls the instance initialization routine. It then executes a message retrieval and dispatch loop that is the top-level control structure for the remainder of execution. The loop is terminated when a `WM_QUIT` message is received, at which time this function exits the application instance by returning the value passed by `PostQuitMessage()`.

If this function must abort before entering the message loop, it returns the conventional value `NULL`.

```
...../
MSG MyMsg = {0};
```

```
MMMain( hInstance, hPrevInstance, lpCmdLine, nCmdShow )
```

```
MSG      msg;
HDC      hdc;
```

```
if (!hPrevInstance)           /* Other instances of app running? */
    if (!InitApplication(hInstance)) /* Initialize shared things */
        return (FALSE);         /* Exits if unable to initialize */
```

/* Perform initializations that apply to a specific instance */

```
if (!InitInstance(hInstance, nCmdShow))
    return (FALSE);
```

```
/* Acquire and dispatch messages until a WM_QUIT message is received. */
while ( GetMessage(&msg, NULL, 0, 0) ) {
    MyMsg = msg;
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

```
)
```

.....

FUNCTION: InitApplication(HANDLE)

PURPOSE: Initializes window data and registers window class

COMMENTS:

This function is called at initialization time only if no other instances of the application are running. This function performs initialization tasks that can be done once for any number of running instances.

In this case, we initialize a window class by filling out a data structure of type WNDCLASS and calling the Windows RegisterClass() function. Since all instances of this application use the same window class, we only need to do this when the first instance is initialized.

.....

```

BOOL InitApplication(hInstance)
HANDLE hInstance;          /* current instance */
{
    WNDCLASS wc;

    /* Fill in window class structure with parameters that describe the
    /* main window. */

    wc.style = CS_OWNDC; /* Class style(s). */
    wc.lpfnWndProc = MainWndProc; /* Function to retrieve messages for */
    /* windows of this class. */
    wc.cbClsExtra = 0; /* No per-class extra data. */
    wc.cbWndExtra = 4; /* No per-window extra data. */
    wc.hInstance = hInstance; /* Application that owns the class. */
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName = "GenericMenu"; /* Name of menu resource in .RC file. */
    wc.lpszClassName = "Generic2Class"; /* Name used in call to CreateWindow. */

    /* Register the window class and return success/failure code. */

    return (RegisterClass(&wc));
}

```

.....

FUNCTION: InitInstance(HANDLE, int)

PURPOSE: Saves instance handle and creates main window

COMMENTS:

This function is called at initialization time for every instance of this application. This function performs initialization tasks that cannot be shared by multiple instances.

In this case, we save the instance handle in a static variable and create and display the main program window.

```

10 ...../
11
12 BOOL InitInstance(hInstance, nCmdShow)
13     HANDLE    hInstance;    /* Current instance identifier. */
14     int       nCmdShow;     /* Param for first ShowWindow() call. */
15 {
16     HWND      hWnd;         /* Main window handle. */
17     HWND      hWndX;
18     OFSTRUCT  ofFileData;
19     HANDLE    hLogFile;
20
21     /* Save the instance handle in static variable, which will be used in */
22     /* many subsequence calls from this application to Windows. */
23
24     hLogFile = OpenFile( "DATA.TXT",
25                         (LPOFSTRUCT)&ofFileData,
26                         OF_CREATE | OF_WRITE | OF_SHARE_DENY_NONE );
27     _lclose( hLogFile );
28
29     hInst = hInstance;
30
31     /* Create a main window for this application instance. */
32
33     hWnd = CreateWindow(
34         "GenericClass",    /* See RegisterClass() call. */
35         "Generic Sample Application", /* Text for window title bar. */
36         WS_OVERLAPPEDWINDOW, /* Window style. */
37         0,                /* Default horizontal position. */
38         0,                /* Default vertical position. */
39         0x280,            /* Default width. */
40         0x1A5,            /* Default height. */
41         NULL,             /* Overlapped windows have no parent. */
42         NULL,             /* Use the window class menu. */
43         hInstance,        /* This instance owns this window. */
44         NULL,             /* Pointer not needed. */
45     );
46     hWndSave = hWnd;
47
48     /* If window could not be created, return "failure" */
49
50     if (!hWnd)
51         return (FALSE);
52
53     /* Make the window visible; update its client area; and return "success" */
54
55     ShowWindow(hWnd, nCmdShow); /* Show the window */
56     UpdateWindow(hWnd);         /* Sends WM_PAINT message */

```

```

return (TRUE);          /* Returns the value from PostQuitMessage */
}

5  /-----/

FUNCTION: MainWndProc(HWND, unsigned, WORD, LONG)

PURPOSE: Processes messages

10 MESSAGES:

WM_COMMAND - application menu (About dialog box)
WM_DESTROY - destroy window

15 COMMENTS:

To process the IDM_ABOUT message, call MakeProcInstance() to get the
current instance address of the About() function. Then call Dialog
box which will create the box according to the information in your
generic.rc file and turn control over to the About() function. When
20 it returns, free the instance address.

-----/

#ifdef WIN16
25 typedef DWORD ULONG;
typedef DWORD BOOLEAN;

typedef struct large_integer {
    ULONG LowPart;
    LONG HighPart;
30 } LARGE_INTEGER, FAR *LPLARGE_INTEGER;

VOID RtlConvertLongToLargeInteger(
    LPLARGE_INTEGER lpli,
    LONG l
35 ) {
    if (l < 0) {
        lpli->HighPart = -l;
    } else {
        lpli->HighPart = 0;
40 }
    lpli->LowPart = (ULONG)l;
}

VOID RtlLargeIntegerSubtract(
    LPLARGE_INTEGER lpliResult,
45 LPLARGE_INTEGER lpliSubtrahend,
    LPLARGE_INTEGER lpliMinuend
) {
    lpliResult->LowPart = lpliSubtrahend->LowPart - lpliMinuend->LowPart;
    if ( (LONG)(lpliResult->LowPart) < 0 ) {
50 lpliResult->HighPart = (lpliSubtrahend->HighPart-1) - lpliMinuend->HighPart;
    } else {
        lpliResult->HighPart = lpliSubtrahend->HighPart - lpliMinuend->HighPart;

```

```

    }
}

5  #define RtlLargeIntegerGreaterThan(X,Y) ( \
    (X).HighPart > (Y).HighPart ? TRUE : \
    (X).HighPart < (Y).HighPart ? FALSE : \
    (X).LowPart > (Y).LowPart ? TRUE : \
    FALSE )

10

#define RTLCONVERTLONGTOLARGEINTEGER( x, y )  RtlConvertLongToLargeInteger( &(x), y )
#define RTLLARGEINTEGERSUBTRACT( x, y, z )    RtlLargeIntegerSubtract( &(x), &(y), &(z) )

15 #else
#define RTLCONVERTLONGTOLARGEINTEGER( x, y )  x = RtlConvertLongToLargeInteger( y )
#define RTLLARGEINTEGERSUBTRACT( x, y, z )    x = RtlLargeIntegerSubtract( y, z )
#endif

20 BOOL fmsgTime = FALSE;
int msgCount = 0;
int msgArray[100] = {0};
LARGE_INTEGER msgTime[100] = {0};
LPOUTLINETEXTMETRIC lp;

25 long FAR PASCAL MainWndProc(HWND, message, wParam, lParam)
HWND hWnd; /* window handle */
UINT message; /* type of message */
WPARAM wParam; /* additional information */
LONG lParam; /* additional information */

30 {
    FARPROC lpProcAbout; /* pointer to the "About" function */
    PAINTSTRUCT ps;
    long rc;
    int i;
    LARGE_INTEGER liFreq;

35 switch (message) {
    case WM_COMMAND: /* message: command from application menu */
        switch( wParam ) {
            case IDM_ABOUT:
                lpProcAbout = MakeProcInstance(About, hInst);

                DialogBox(hInst, /* current instance */
                    "AboutBox", /* resource to use */
                    hWnd, /* parent handle */
                    lpProcAbout); /* About() instance address */

40
                FreeProcInstance(lpProcAbout);
                break; /* Lets Windows process it */
            case IDM_DIV0:
                {
                    int i, j;
                    char text[50];
                    i = 5;

50
                }
            }
        }
}

```

```

        while ( i >= 0 ) {
            j = 360/i;
            wsprintf(text, "I = %d, j = %d\n", i, j);
            -i;
5          }
        }
        break;
default:
        return (DefWindowProc(hWnd, message, wParam, lParam));
10    }

case WM_DESTROY:          /* message: window being destroyed */

        PostQuitMessage(0);
        break;
15    case WM_PAINT:
        {
            WNDCLASS wc;
            LPSTR lp;
            HDC hDC;
            PAINTSTRUCT ps;
            char text[100];
            HWND hwnd2;
            ATOM Atom;
            int rc;
            HBRUSH hbr;
            LPSTR lpstr;
            int x;
            int y;

            hDC = BeginPaint( hWnd, &ps );

            lpstr = (LPSTR)"Hi There";

            hbr = CreateSolidBrush( RGB(0x30,0x30,0) );
            GrayString( hDC, hbr, (GRAYSTRINGPROC)NULL, (LPARAM)lpstr, 0, 10, 10, 0, 0 );
            SetBkMode( hDC, TRANSPARENT );
            SetTextColor( hDC, RGB(0,0,0) );

            x = 150; y = 150;

            TextOut( hDC, x-1, y-1, "Testing 2", 9 );
            TextOut( hDC, x, y-1, "Testing 2", 9 );
            TextOut( hDC, x+1, y-1, "Testing 2", 9 );
            TextOut( hDC, x-1, y, "Testing 2", 9 );
            TextOut( hDC, x+1, y, "Testing 2", 9 );
            TextOut( hDC, x-1, y+1, "Testing 2", 9 );
            TextOut( hDC, x, y+1, "Testing 2", 9 );
            TextOut( hDC, x+1, y+1, "Testing 2", 9 );

            SetTextColor( hDC, RGB(0xFF,0xFF,0xFF) );
            TextOut( hDC, x, y, "Testing 2", 9 );

            EndPaint( hWnd, &ps );
55

```

```

    }
    break;

case WM_LBUTTONDOWN:
5   {
        unsigned short i;
        unsigned short j;

        j = 1;
        while (j <= 200) {
10         i = 1;
            while (i) {
                _asm { mov dx,ax };

                i++;
15         }
                j++;
            }
        }

    default: /* Passes it on if unprocessed */
20     rc = DefWindowProc(hWnd, message, wParam, lParam);
        return( rc );
    }
    return (NULL);
25 }

.....

FUNCTION: About(HWND, unsigned, WPARAM, LONG)
30
PURPOSE: Processes messages for "About" dialog box

MESSAGES:

WM_INITDIALOG - initialize dialog box
35 WM_COMMAND   - Input received

COMMENTS:

No initialization is needed for this particular dialog box, but TRUE
40 must be returned to Windows.

Wait for user to click on "Ok" button, then close the dialog box.

...../

45 BOOL FAR PASCAL About(hDlg, message, wParam, lParam)
    HWND hDlg; /* window handle of the dialog box */
    UINT message; /* type of message */
    WPARAM wParam; /* message-specific information */
    LONG lParam;
50 {

```

55

```

switch (message) {
    case WM_INITDIALOG:          /* message: initialize dialog box */
        return (TRUE);

5      case WM_COMMAND:          /* message: received a command */
        if (wParam == IDOK      /* "OK" box selected? */
            || wParam == IDCANCEL) { /* System menu close command? */
            EndDialog(hDlg, TRUE); /* Exits the dialog box */
            return (TRUE);
        }
10      break;
    }
    return (FALSE);              /* Didn't process a message */
}

```

TABLE 2

LOG FILE

```

20  1  011842053E61APICALL:RegisterClass {20 0B3F0208 0 4 B1E B6E 19E 5C "GenericMenu"
    "Generic2Class" 0 0A1F0117
2  01184206B9B1APIRET:RegisterClass 003E
3  01184206EFA1APICALL:OpenFile "TIME.LOG" {2 1A B1F D7 A7 0 0 } 1041
4  0118420CF031APIRET:OpenFile 5 {21 1 0 C2 1A F8 71 44 3A 5C 54 45 53 54 53 5C 47
    45 4E 45 52 49 43 EC 54 49 4D 45 2E 4C 4F 47 0 }
25  5  0118420DAED1APICALL:_lclose 5
    6  0118420E5701APIRET:_lclose 0
    7  011842102B31APICALL:CreateWindow "Generic2Class" "Generic Sample Application"
        CF0000 0 0 280 1A5 0 0 B1E 0
    9  021842149C51MSGCALL:0B3F0208 3A 24-WM_GETMINMAXINFO 0 {24 24} {408 308} {FFFC
        FFFC} {66 1A} {408 308}
30  9  03184214FBC1APICALL:DefWindowProc 3A 24-WM_GETMINMAXINFO 0 {24 24} {408 308} {FFFC
        FFFC} {66 1A} {408 308}
    10 031842155B01APIRET:DefWindowProc 0
    11 021842158D61MSGRET:0 0B3F0208 3A 24-WM_GETMINMAXINFO 0 {24 24} {408 308} {FFFC
        FFFC} {66 1A} {408 308}
35  12 0218421662B1MSGCALL:0B3F0208 3A 31-WM_NCCREATE 0 {0 B1E 99 0 1A5 280 0 0 CF0000
    "Generic Sample Application" "Generic2Class" 0}
    13 03184216EA31APICALL:DefWindowProc 3A 31-WM_NCCREATE 0 {0 B1E 99 0 1A5 280 0 0
        CF0000 "Generic Sample Application" "Generic2Class" 0}
    14 03184217B111APIRET:DefWindowProc 1
    15 02184217E491MSGRET:1 0B3F0208 3A 31-WM_NCCREATE 0 {0 B1E 99 0 1A5 280 0 0 CF0000
        "Generic Sample Application" "Generic2Class" 0}
40  16 021842188CF1MSGCALL:0B3F0208 3A 33-WM_NCCALCSIZE 0 {0 0 280 1A5}
    17 03184218CC71APICALL:DefWindowProc 3A 33-WM_NCCALCSIZE 0 {0 0 280 1A5}
    18 031842199AD1APIRET:DefWindowProc 0
    19 02184219CED1MSGRET:0 0B3F0208 3A 33-WM_NCCALCSIZE 0 {4 2A 27C 1A1}
45  20 0218421A4DB1MSGCALL:0B3F0208 3A 1-WM_CREATE 0 {0 B1E 99 0 1A5 280 0 0 CF0000
    "Generic Sample Application" "Generic2Class" 0}
    21 0318421AC7F1APICALL:DefWindowProc 3A 1-WM_CREATE 0 {0 B1E 99 0 1A5 280 0 0 CF0000
        "Generic Sample Application" "Generic2Class" 0}
    22 0318421B49C1APIRET:DefWindowProc 0
    23 0218421B7AE1MSGRET:0 0B3F0208 3A 1-WM_CREATE 0 {0 B1E 99 0 1A5 280 0 0 CF0000
        "Generic Sample Application" "Generic2Class" 0}
50  24 0118421C18A1APIRET:CreateWindow 3A {0 0 280 1A5} {0 0 280 1A5}

```

EP 0 632 377 A1

25 0118421C541:APICALL:ShowWindow 3A 1
 26 0218421C03:MSGCALL:0B3F0208 3A 18-WM_SHOWWINDOW 1 0
 27 0318421CF54:APICALL:DefWindowProc 3A 18-WM_SHOWWINDOW 1 0
 28 0318421D383:APIRET:DefWindowProc 0
 5 29 0218421D69B:MSGRET:0 0B3F0208 3A 18-WM_SHOWWINDOW 1 0
 30 0218421E196:MSGCALL:0B3F0208 3A 46-WM_WINDOWPOSCHANGING 0 C370180
 31 0318421E55E:APICALL:DefWindowProc 3A 46-WM_WINDOWPOSCHANGING 0 C370180
 32 0318421EABB:APIRET:DefWindowProc 0
 33 0218421EDE5:MSGRET:0 0B3F0208 3A 46-WM_WINDOWPOSCHANGING 0 C370180
 10 34 0218422093F:MSGCALL:0B3F0208 3A 30F-WM_QUERYNEWPALETTE 0 0
 35 03184220D55:APICALL:DefWindowProc 3A 30F-WM_QUERYNEWPALETTE 0 0
 36 03184221483:APIRET:DefWindowProc 0
 37 021842217AB:MSGRET:0 0B3F0208 3A 30F-WM_QUERYNEWPALETTE 0 0
 38 02184221F6C:MSGCALL:0B3F0208 3A 46-WM_WINDOWPOSCHANGING 0 C370180
 39 03184222313:APICALL:DefWindowProc 3A 46-WM_WINDOWPOSCHANGING 0 C370180
 15 40 031842227BC:APIRET:DefWindowProc 0
 41 02184222AD7:MSGRET:0 0B3F0208 3A 46-WM_WINDOWPOSCHANGING 0 C370180
 42 02184223361:MSGCALL:0B3F0208 3A 1C-WM_ACTIVATEAPP 1 0 0
 43 0318422370A:APICALL:DefWindowProc 3A 1C-WM_ACTIVATEAPP 1 0 0
 44 03184223AEF:APIRET:DefWindowProc 0
 20 45 02184223DF4:MSGRET:0 0B3F0208 3A 1C-WM_ACTIVATEAPP 1 0 0
 46 02184224454:MSGCALL:0B3F0208 3A 36-WM_NCACTIVATE 1 0
 47 031842247BA:APICALL:DefWindowProc 3A 36-WM_NCACTIVATE 1 0
 48 0418422576D:MSGCALL:0B3F0208 3A 3D-WM_GETTEXT 4F 0C370180
 49 05184225B46:APICALL:DefWindowProc 3A 3D-WM_GETTEXT 4F 0C370180
 50 0518422627D:APIRET:DefWindowProc 1A
 25 51 041842265B4:MSGRET:1A 0B3F0208 3A 3D-WM_GETTEXT 4F "Generic Sample Application"
 52 03184227DBE:APIRET:DefWindowProc 1
 53 02184229349:MSGRET:1 0B3F0208 3A 36-WM_NCACTIVATE 1 0
 54 02184229C98:MSGCALL:0B3F0208 3A 6-WM_ACTIVATE 1 0 0
 55 03184229017:APICALL:DefWindowProc 3A 6-WM_ACTIVATE 1 0 0
 30 56 0418422971B:MSGCALL:0B3F0208 3A 7-WM_SETFOCUS 0 0
 57 05184229A78:APICALL:DefWindowProc 3A 7-WM_SETFOCUS 0 0
 58 0518422A23C:APIRET:DefWindowProc 0
 59 0418422A556:MSGRET:0 0B3F0208 3A 7-WM_SETFOCUS 0 0
 60 0318422C408:APIRET:DefWindowProc 0
 61 0218422C932:MSGRET:0 0B3F0208 3A 6-WM_ACTIVATE 1 0 0
 35 62 0218422D03E:MSGCALL:0B3F0208 3A 35-WM_NCPAINT E4 0
 63 0318422D3A8:APICALL:DefWindowProc 3A 35-WM_NCPAINT E4 0
 64 0418422F419:MSGCALL:0B3F0208 3A 3D-WM_GETTEXT 4F 0C370180
 65 0518422F7F0:APICALL:DefWindowProc 3A 3D-WM_GETTEXT 4F 0C370180
 66 0518422FCBB:APIRET:DefWindowProc 1A
 40 67 0418423063A:MSGRET:1A 0B3F0208 3A 3D-WM_GETTEXT 4F "Generic Sample Application"
 68 03184231791:APIRET:DefWindowProc 0
 69 02184231ADA:MSGRET:0 0B3F0208 3A 35-WM_NCPAINT E4 0
 70 02184232218:MSGCALL:0B3F0208 3A 14-WM_ERASEBKGD E4 0
 71 03184232599:APICALL:DefWindowProc 3A 14-WM_ERASEBKGD E4 0
 72 031842331A7:APIRET:DefWindowProc 1
 45 73 021842334E8:MSGRET:1 0B3F0208 3A 14-WM_ERASEBKGD E4 0
 74 02184233D1B:MSGCALL:0B3F0208 3A 47-WM_WINDOWPOSCHANGED 0 C370180
 75 031842340CC:APICALL:DefWindowProc 3A 47-WM_WINDOWPOSCHANGED 0 C370180
 76 03184234590:APIRET:DefWindowProc 0
 77 0214235DB8:MSGRET:0 0B3F0208 3A 47-WM_WINDOWPOSCHANGED 0 C370180
 50 78 0218423651A:MSGCALL:0B3F0208 3A 5-WM_SIZE 3 1770278
 79 03184236865:APICALL:DefWindowProc 3A 5-WM_SIZE 0 1770278
 80 03184236C33:APIRET:DefWindowProc 0

```

31 02184236F3B1MSGRET:0 0B3F0208 3A 5-WM_SIZE 0 1770278
32 0218423746A1MSGCALL:0B3F0208 3A 3-WM_MOVE 0 2A0004
33 031842377961APICALL:DefWindowProc 3A 3-WM_MOVE 0 2A0004
34 03184237B3E1APIRET:DefWindowProc 0
5 02184237E441MSGRET:0 0B3F0208 3A 3-WM_MOVE 0 2A0004
36 031842383241APIRET:ShowWindow FALSE
37 0118423860D1APICALL:UpdateWindow 8A
38 02184238C351MSGCALL:0B3F0208 3A F-WM_PAINT 0 0
39 03184238FEC1APICALL:BeginPaint 9A
10 031842397061APIRET:BeginPaint E4 {E4 FALSE (0 0 278 177) FALSE FALSE 17 1 1F 8A
3A 0 FF FF 14 0 0 0 95 5 9E 0 }
91 0318423A21C1APICALL:CreateSolidBrush 3030
92 0318423B05D1APIRET:CreateSolidBrush F0
93 0318423B3A51APICALL:GrayString E4 F0 NULL (80 0 B1F0473 0 A A 0 0
94 0318423C2E31APIRET:GrayString TRUE
15 0318423C6431APICALL:SetBkMode E4 1
96 0318423CD831APIRET:SetBkMode 2
97 0318423EE061APICALL:SetTextColor E4 0
98 0318423F4B91APIRET:SetTextColor 0
99 0318423FB391APICALL:TextOut E4 95 95 "Testing 2" 9
20 031842400651APIRET:TextOut TRUE
101 031842403491APICALL:TextOut E4 96 95 "Testing 2" 9
102 03184240AFO1APIRET:TextOut TRUE
103 03184240DD41APICALL:TextOut E4 97 95 "Testing 2" 9
104 031842412881APIRET:TextOut TRUE
105 0318424155B1APICALL:TextOut E4 95 96 "Testing 2" 9
25 106 03184241D701APIRET:TextOut TRUE
107 031842420601APICALL:TextOut E4 97 96 "Testing 2" 9
108 031842425251APIRET:TextOut TRUE
109 031842427FB1APICALL:TextOut E4 95 97 "Testing 2" 9
110 03184242CAC1APIRET:TextOut TRUE
30 111 03184242F911APICALL:TextOut E4 96 97 "Testing 2" 9
112 031842492301APIRET:TextOut TRUE
113 031842495711APICALL:TextOut E4 97 97 "Testing 2" 9
114 03184249A961APIRET:TextOut TRUE
115 03184249D6E1APICALL:SetTextColor E4 FFFFFFFF
116 0318424A1E61APIRET:SetTextColor 0
35 117 0318424A57D1APICALL:TextOut E4 96 96 "Testing 2" 9
118 0318424AA391APIRET:TextOut TRUE
119 0318424AD191APICALL:EndPaint 3A {E4 FALSE (0 0 278 177) FALSE FALSE 17 1 1F 8A 8A
0 FF FF 14 0 0 0 95 5 9E 0 }
120 0318424C06A1APIRET:EndPaint
40 121 0218424C2FC1MSGRET:0 0B3F0208 3A F-WM_PAINT 0 0
122 0118424C9331APIRET:UpdateWindow
123 0118424CADB1APICALL:GetMessage 0 0 0
124 0218424D2A61MSGCALL:0B3F0208 8A 84-WM_NCHITTEST 0 8700E7
125 0318424D63E1APICALL:DefWindowProc 8A 84-WM_NCHITTEST 0 8700E7
126 0318424DAF71APIRET:DefWindowProc 1
45 127 0218424DE211MSGRET:1 0B3F0208 8A 84-WM_NCHITTEST 0 8700E7
128 0218424E7E61MSGCALL:0B3F0208 9A 20-WM_SETCURSOR 8A 2000001
129 0318424EC7D1APICALL:DefWindowProc 9A 20-WM_SETCURSOR 8A 2000001
130 0318424F7E91APIRET:DefWindowProc 0
131 0218424FB291MSGRET:0 0B3F0208 3A 20-WM_SETCURSOR 8A 2000001
132 011842501D61APIRET:GetMessage TRUE (8A 200 0 5D00E3 001C5B1F (E7 87) )
50 133 0118425080F1APICALL:TranslateMessage (8A 200 0 5D00E3 001C5B1F (E7 87) )
134 01184250D971APIRET:TranslateMessage FALSE

```

EP 0 632 377 A1

135 018425107A:APICALL:DispatchMessage (8A 200 0 5D00E3 001C5B1F (E7 97))
 136 021842516AE:MSGCALL:0B3F0208 8A 200-WM_MOUSEMOVE/WM_MOUSEFIRST 0 5D00E3
 137 03184251A8B:APICALL:DefWindowProc 8A 200-WM_MOUSEMOVE/WM_MOUSEFIRST 0 5D00E3
 138 03184251EC6:APIRET:DefWindowProc 0
 5 139 021842521D0:MSGRET:0 0B3F0208 8A 200-WM_MOUSEMOVE/WM_MOUSEFIRST 0 5D00E3
 140 01842525BD:APIRET:DispatchMessage 0
 141 01842528B0:APICALL:GetMessage 0 0 0
 142 0218431B873:MSGCALL:0B3F0208 8A 84-WM_NCHITTEST 0 8700E6
 143 0318431C5C6:APICALL:DefWindowProc 8A 84-WM_NCHITTEST 0 8700E6
 144 0318431C883:APIRET:DefWindowProc 1
 10 145 0218431CEC0:MSGRET:1 0B3F0208 8A 84-WM_NCHITTEST 0 8700E6
 146 0218431D5B9:MSGCALL:0B3F0208 8A 20-WM_SETCURSOR 8A 2000001
 147 0318431D959:APICALL:DefWindowProc 8A 20-WM_SETCURSOR 8A 2000001
 148 0318431DFA7:APIRET:DefWindowProc 0
 149 0218431E2D3:MSGRET:0 0B3F0208 8A 20-WM_SETCURSOR 8A 2000001
 15 150 018431E9821A:APIRET:GetMessage TRUE (8A 200 0 5D00E2 001C5DE0 (E6 87))
 151 018431EDB4:APICALL:TranslateMessage (8A 200 0 5D00E2 001C5DE0 (E6 87))
 152 018431FC2D:APIRET:TranslateMessage FALSE
 153 018431FF1E:APICALL:DispatchMessage (8A 200 0 5D00E2 001C5DE0 (E6 87))
 154 021843205FB:MSGCALL:0B3F0208 8A 200-WM_MOUSEMOVE/WM_MOUSEFIRST 0 5D00E2
 155 031843209DF:APICALL:DefWindowProc 8A 200-WM_MOUSEMOVE/WM_MOUSEFIRST 0 5D00E2
 20 156 03184320E18:APIRET:DefWindowProc 0
 157 02184321129:MSGRET:0 0B3F0208 8A 200-WM_MOUSEMOVE/WM_MOUSEFIRST 0 5D00E2
 158 01843219D0:APIRET:DispatchMessage 0
 159 0184321CFD:APICALL:GetMessage 0 0 0
 160 02184322475:MSGCALL:0B3F0208 8A 84-WM_NCHITTEST 0 8700E5
 25 161 03184322814:APICALL:DefWindowProc 8A 84-WM_NCHITTEST 0 8700E5
 162 03184322C9F:APIRET:DefWindowProc 1
 163 021843232FE:MSGRET:1 0B3F0208 8A 84-WM_NCHITTEST 0 8700E5
 164 02184323C29:MSGCALL:0B3F0208 8A 20-WM_SETCURSOR 8A 2000001
 165 0318432402D:APICALL:DefWindowProc 8A 20-WM_SETCURSOR 8A 2000001
 166 03184324658:APIRET:DefWindowProc 0
 30 167 02184324984:MSGRET:0 0B3F0208 8A 20-WM_SETCURSOR 8A 2000001
 168 01843250D1:APIRET:GetMessage TRUE (8A 200 0 5D00E1 001C5DFE (E5 87)) ???
 169 0184325808:APICALL:TranslateMessage (8A 200 0 5D00E1 001C5DFE (E5 87))
 170 0184325D8A:APIRET:TranslateMessage FALSE
 171 018432606C:APICALL:DispatchMessage (8A 200 0 5D00E1 001C5DFE (E5 87))
 35 172 02184326F02:MSGCALL:0B3F0208 8A 200-WM_MOUSEMOVE/WM_MOUSEFIRST 0 5D00E1
 173 031843272ED:APICALL:DefWindowProc 8A 200-WM_MOUSEMOVE/WM_MOUSEFIRST 0 5D00E1
 174 0318432772D:APIRET:DefWindowProc 0
 175 02184327A3D:MSGRET:0 0B3F0208 8A 200-WM_MOUSEMOVE/WM_MOUSEFIRST 0 5D00E1
 176 0184327E32:APIRET:DispatchMessage 0
 177 018442369A:APICALL:GetMessage 0 0 0
 40 178 018456DF3E:APIRET:GetMessage TRUE (8A 100 10 360001 001C65D8 (12E 7F))
 179 018456E4D9:APICALL:TranslateMessage (8A 100 10 360001 001C65D8 (12E 7F))
 180 018456ED61:APIRET:TranslateMessage TRUE
 181 018456FOA2:APICALL:DispatchMessage (8A 100 10 360001 001C65D8 (12E 7F))
 182 0218456FA00:MSGCALL:0B3F0208 8A 100-WM_KEYDOWN/WM_KEYFIRST 10 360001
 45 183 0318456FECA:APICALL:DefWindowProc 8A 100-WM_KEYDOWN/WM_KEYFIRST 10 360001
 184 0318457041C:APIRET:DefWindowProc 0
 185 0218457077A:MSGRET:0 0B3F0208 8A 100-WM_KEYDOWN/WM_KEYFIRST 10 360001
 186 0184570BD3:APIRET:DispatchMessage 0
 187 0184570F03:APICALL:GetMessage 0 0 0
 188 018459F38C:APIRET:GetMessage TRUE (8A 100 48 230001 001C668C (12E 7F))
 50 189 018459F850:APICALL:TranslateMessage (8A 100 48 230001 001C668C (12E 7F))
 190 01845A013A:APIRET:TranslateMessage TRUE

191 011845A04751APICALL:DispatchMessage (8A 100 48 230001 001C668C (12E 7F))
 192 021845A0B381MSGCALL:0B3F0208 3A 100-WM_KEYDOWN/WM_KEYFIRST 48 230001
 193 031845A0F7B1APICALL:DefWindowProc 3A 100-WM_KEYDOWN/WM_KEYFIRST 48 230001
 194 031845A17C51APIRET:DefWindowProc 0
 5 195 021845A1B501MSGRET:0 0B3F0208 3A 100-WM_KEYDOWN/WM_KEYFIRST 48 230001
 196 011845A1FBD1APIRET:DispatchMessage 0
 197 011845A22EC1APICALL:GetMessage 0 0 0
 198 011845A342A1APIRET:GetMessage TRUE (8A 102 48 230001 001C668C (12E 7F))
 199 011845A38D61APICALL:TranslateMessage (8A 102 48 230001 001C668C (12E 7F))
 200 011845A3ED01APIRET:TranslateMessage FALSE
 10 201 011845A41F11APICALL:DispatchMessage (8A 102 48 230001 001C668C (12E 7F))
 202 021845A48AB1MSGCALL:0B3F0208 8A 102-WM_CHAR 48 230001
 203 031845A4C8F1APICALL:DefWindowProc 8A 102-WM_CHAR 48 230001
 204 031845A50D21APIRET:DefWindowProc 0
 205 021845A541A1MSGRET:0 0B3F0208 3A 102-WM_CHAR 48 230001
 206 011845A58D01APIRET:DispatchMessage 0
 15 207 011845A5B391APICALL:GetMessage 0 0 0
 208 011845BACF61APIRET:GetMessage TRUE (8A 101 48 C0230001 001C66E6 (12E 7F))
 209 011845BB1D41APICALL:TranslateMessage (8A 101 48 C0230001 001C66E6 (12E 7F))
 210 011845BB9D91APIRET:TranslateMessage TRUE
 211 011845BBD141APICALL:DispatchMessage (8A 101 48 C0230001 001C66E6 (12E 7F))
 20 212 021845BC3F31MSGCALL:0B3F0208 3A 101-WM_KEYUP 48 C0230001
 213 031845BC9041APICALL:DefWindowProc 3A 101-WM_KEYUP 48 C0230001
 214 031845BCFA81APIRET:DefWindowProc 0
 215 021845BD3251MSGRET:0 0B3F0208 3A 101-WM_KEYUP 48 C0230001
 216 011845BD7461APIRET:DispatchMessage 0
 217 011845BDA7A1APICALL:GetMessage 0 0 0
 25 218 011845D0A5E1APIRET:GetMessage TRUE (8A 101 10 C0360001 001C6731 (12E 7F))
 219 011845D0ECS1APICALL:TranslateMessage (8A 101 10 C0360001 001C6731 (12E 7F))
 220 011845D138B1APIRET:TranslateMessage TRUE
 221 011845D1E9C1APICALL:DispatchMessage (8A 101 10 C0360001 001C6731 (12E 7F))
 222 021845D24F51MSGCALL:0B3F0208 3A 101-WM_KEYUP 10 C0360001
 30 223 031845D28AD1APICALL:DefWindowProc 3A 101-WM_KEYUP 10 C0360001
 224 031845D33081APIRET:DefWindowProc 0
 225 021845D36461MSGRET:0 0B3F0208 3A 101-WM_KEYUP 10 C0360001
 226 011845D3A0E1APIRET:DispatchMessage 0
 227 011845D3D041APICALL:GetMessage 0 0 0
 228 011845F5E5A1APIRET:GetMessage TRUE (8A 100 45 120001 001C67A9 (12E 7F))
 35 229 011845F62AF1APICALL:TranslateMessage (8A 100 45 120001 001C67A9 (12E 7F))
 230 011845F6B381APIRET:TranslateMessage TRUE
 231 011845F6E2E1APICALL:DispatchMessage (8A 100 45 120001 001C67A9 (12E 7F))
 232 021845F748B1MSGCALL:0B3F0208 3A 100-WM_KEYDOWN/WM_KEYFIRST 45 120001
 233 031845F78731APICALL:DefWindowProc 3A 100-WM_KEYDOWN/WM_KEYFIRST 45 120001
 40 234 031845F7D431APIRET:DefWindowProc 0
 235 021845F80611MSGRET:0 0B3F0208 3A 100-WM_KEYDOWN/WM_KEYFIRST 45 120001
 236 011845F84481APIRET:DispatchMessage 0
 237 011845F87381APICALL:GetMessage 0 0 0
 238 011845F8D361APIRET:GetMessage TRUE (8A 102 65 120001 001C67B8 (12E 7F))
 239 011845F913A1APICALL:TranslateMessage (8A 102 65 120001 001C67B8 (12E 7F))
 45 240 011845F96B01APIRET:TranslateMessage FALSE
 241 011845F99901APICALL:DispatchMessage (8A 102 65 120001 001C67B8 (12E 7F))
 242 021845FB0AF1MSGCALL:0B3F0208 3A 102-WM_CHAR 65 120001
 243 031845FB47D1APICALL:DefWindowProc 3A 102-WM_CHAR 65 120001
 244 031845FB86B1APIRET:DefWindowProc 0
 50 245 021845FB8BC1MSGRET:0 0B3F0208 3A 102-WM_CHAR 65 120001
 246 011845FBFCF1APIRET:DispatchMessage 0

247 011845FC21C:APICALL:GetMessage 0 0 0
 248 0118460E531:APIRET:GetMessage TRUE (8A 101 45 C0120001 001C6803 (12E 7F))
 249 0118460E999:APICALL:TranslateMessage (8A 101 45 C0120001 001C6803 (12E 7F))
 250 0118460F436:APIRET:TranslateMessage TRUE
 251 0118460F737:APICALL:DispatchMessage (8A 101 45 C0120001 001C6803 (12E 7F))
 252 0218460FE34:MSGCALL:0B3F0208 8A 101-WM_KEYUP 45 C0120001
 253 031846101EE:APICALL:DefWindowProc 9A 101-WM_KEYUP 45 C0120001
 254 03184610841:APIRET:DefWindowProc 0
 255 02184610B7D:MSGRET:0 0B3F0208 8A 101-WM_KEYUP 45 C0120001
 256 01184610F3C:APIRET:DispatchMessage 0
 257 01184611231:APICALL:GetMessage 0 0 0
 258 011846513E3:APIRET:GetMessage TRUE (8A 100 4C 260001 001C68E4 (12E 7F))
 259 011846519DC:APICALL:TranslateMessage (8A 100 4C 260001 001C68E4 (12E 7F))
 260 01184652174:APIRET:TranslateMessage TRUE
 261 0118465246A:APICALL:DispatchMessage (8A 100 4C 260001 001C68E4 (12E 7F))
 262 02184652ADI:MSGCALL:0B3F0208 8A 100-WM_KEYDOWN/WM_KEYFIRST 4C 260001
 263 03184652EB9:APICALL:DefWindowProc 9A 100-WM_KEYDOWN/WM_KEYFIRST 4C 260001
 264 0318465338D:APIRET:DefWindowProc 0
 265 021846536A9:MSGRET:0 0B3F0208 8A 100-WM_KEYDOWN/WM_KEYFIRST 4C 260001
 266 01184653A91:APIRET:DispatchMessage 0
 267 01184653D82:APICALL:GetMessage 0 0 0
 268 01184654393:APIRET:GetMessage TRUE (8A 102 6C 260001 001C68F3 (12E 7F))
 269 01184654796:APICALL:TranslateMessage (8A 102 6C 260001 001C68F3 (12E 7F))
 270 01184654D0C:APIRET:TranslateMessage FALSE
 271 01184654FEC:APICALL:DispatchMessage (8A 102 6C 260001 001C68F3 (12E 7F))
 272 02184655604:MSGCALL:0B3F0208 8A 102-WM_CHAR 6C 260001
 273 03184655978:APICALL:DefWindowProc 9A 102-WM_CHAR 6C 260001
 274 031846570A4:APIRET:DefWindowProc 0
 275 021846573D4:MSGRET:0 0B3F0208 8A 102-WM_CHAR 6C 260001
 276 0118465779A:APIRET:DispatchMessage 0
 277 01184657A7D:APICALL:GetMessage 0 0 0
 278 01184661F68:APIRET:GetMessage TRUE (8A 101 4C C0260001 001C6920 (12E 7F))
 279 011846623B6:APICALL:TranslateMessage (8A 101 4C C0260001 001C6920 (12E 7F))
 280 01184662B62:APIRET:TranslateMessage TRUE
 281 01184663E6E:APICALL:DispatchMessage (8A 101 4C C0260001 001C6920 (12E 7F))
 282 02184664790:MSGCALL:0B3F0208 8A 101-WM_KEYUP 4C C0260001
 283 03184664B3E:APICALL:DefWindowProc 9A 101-WM_KEYUP 4C C0260001
 284 031846651D1:APIRET:DefWindowProc 0
 285 02184665509:MSGRET:0 0B3F0208 8A 101-WM_KEYUP 4C C0260001
 286 011846658C7:APIRET:DispatchMessage 0
 287 01184665BBD:APICALL:GetMessage 0 0 0
 288 01184678FDO:APIRET:GetMessage TRUE (8A 100 4C 260001 001C696B (12E 7F))
 289 01184679422:APICALL:TranslateMessage (8A 100 4C 260001 001C696B (12E 7F))
 290 01184679F3D:APIRET:TranslateMessage TRUE
 291 0118467A23F:APICALL:DispatchMessage (8A 100 4C 260001 001C696B (12E 7F))
 292 0218467A8AE:MSGCALL:0B3F0208 8A 100-WM_KEYDOWN/WM_KEYFIRST 4C 260001
 293 0318467AC99:APICALL:DefWindowProc 9A 100-WM_KEYDOWN/WM_KEYFIRST 4C 260001
 294 0318467B163:APIRET:DefWindowProc 0
 295 0218467B483:MSGRET:0 0B3F0208 8A 100-WM_KEYDOWN/WM_KEYFIRST 4C 260001
 296 0118467B868:APIRET:DispatchMessage 0
 297 0118467BB57:APICALL:GetMessage 0 0 0
 298 0118467C16E:APIRET:GetMessage TRUE (8A 102 6C 260001 001C697A (12E 7F))
 299 0118467C573:APICALL:TranslateMessage (8A 102 6C 260001 001C697A (12E 7F))
 300 0118467CAEC:APIRET:TranslateMessage FALSE
 301 0118467CDCB:APICALL:DispatchMessage (8A 102 6C 260001 001C697A (12E 7F))
 302 0218467E95C:MSGCALL:0B3F0208 8A 102-WM_CHAR 6C 260001

203 0318467ED041APICALL:DefWindowProc 9A 102-WM_CHAR 6C 260001
 204 0318467F0F31APIRET:DefWindowProc 0
 205 0218467F4031MSGRET:0 0B3F0208 9A 102-WM_CHAR 6C 260001
 5 206 0118467F79F1APIRET:DispatchMessage 0
 207 0118467FAB01APICALL:GetMessage 0 0 0
 208 0118468BE511APIRET:GetMessage TRUE (8A 101 4C C0260001 001C69B6 (12E 7F))
 209 0118468CC29D1APICALL:TranslateMessage (8A 101 4C C0260001 001C69B6 (12E 7F))
 210 0118468CA521APIRET:TranslateMessage TRUE
 211 0118468CD491APICALL:DispatchMessage (8A 101 4C C0260001 001C69B6 (12E 7F))
 10 212 0218468D6E51MSGCALL:0B3F0208 9A 101-WM_KEYUP 4C C0260001
 213 0318468DA891APICALL:DefWindowProc 9A 101-WM_KEYUP 4C C0260001
 214 0318468E10C1APIRET:DefWindowProc 0
 215 0218468E4431MSGRET:0 0B3F0208 9A 101-WM_KEYUP 4C C0260001
 216 0118468E8AC1APIRET:DispatchMessage 0
 217 0118468EBA61APICALL:GetMessage 0 0 0
 15 218 011846A798F1APIRET:GetMessage TRUE (8A 100 4F 180001 001C6A10 (12E 7F))
 219 011846A7DC91APICALL:TranslateMessage (8A 100 4F 180001 001C6A10 (12E 7F))
 220 011846A863C1APIRET:TranslateMessage TRUE
 221 011846A89321APICALL:DispatchMessage (8A 100 4F 180001 001C6A10 (12E 7F))
 222 021846A906E1MSGCALL:0B3F0208 9A 100-WM_KEYDOWN/WM_KEYFIRST 4F 180001
 20 223 031846A943B1APICALL:DefWindowProc 9A 100-WM_KEYDOWN/WM_KEYFIRST 4F 180001
 224 031846A98F61APIRET:DefWindowProc 0
 225 021846A9C121MSGRET:0 0B3F0208 9A 100-WM_KEYDOWN/WM_KEYFIRST 4F 180001
 226 011846A9FF21APIRET:DispatchMessage 0
 227 011846AA2E21APICALL:GetMessage 0 0 0
 25 228 011846AACB1APIRET:GetMessage TRUE (8A 102 6F 180001 001C6A10 (12E 7F))
 229 011846AB0C51APICALL:TranslateMessage (8A 102 6F 180001 001C6A10 (12E 7F))
 230 011846AB63F1APIRET:TranslateMessage FALSE
 231 011846AB91D1APICALL:DispatchMessage (8A 102 6F 180001 001C6A10 (12E 7F))
 232 021846ABF481MSGCALL:0B3F0208 9A 102-WM_CHAR 6F 180001
 233 031846AC6421APICALL:DefWindowProc 9A 102-WM_CHAR 6F 180001
 30 234 031846ACA311APIRET:DefWindowProc 0
 235 021846ACD3C1MSGRET:0 0B3F0208 9A 102-WM_CHAR 6F 180001
 236 011846AD16D1APIRET:DispatchMessage 0
 237 011846AD4601APICALL:GetMessage 0 0 0
 238 011846BBE041APIRET:GetMessage TRUE (8A 101 4F C0180001 001C6A5B (12E 7F))
 239 011846BC24E1APICALL:TranslateMessage (8A 101 4F C0180001 001C6A5B (12E 7F))
 35 240 011846BC9C11APIRET:TranslateMessage TRUE
 241 011846BCCB61APICALL:DispatchMessage (8A 101 4F C0180001 001C6A5B (12E 7F))
 242 021846BD2FC1MSGCALL:0B3F0208 9A 101-WM_KEYUP 4F C0180001
 243 031846BD69E1APICALL:DefWindowProc 9A 101-WM_KEYUP 4F C0180001
 244 031846BDCC51APIRET:DefWindowProc 0
 245 021846BDDFF1MSGRET:0 0B3F0208 9A 101-WM_KEYUP 4F C0180001
 40 246 011846BE3B61APIRET:DispatchMessage 0
 247 011846BE6AD1APICALL:GetMessage 0 0 0
 248 0118485206D1APIRET:GetMessage TRUE (8A 104 12 20380001 001C6FCE (12E 7F))
 249 011848524E01APICALL:TranslateMessage (8A 104 12 20380001 001C6FCE (12E 7F))
 250 01184852CEB1APIRET:TranslateMessage TRUE
 45 251 01184852FE31APICALL:DispatchMessage (8A 104 12 20380001 001C6FCE (12E 7F))
 252 0218485365B1MSGCALL:0B3F0208 9A 104-WM_SYSKEYDOWN 12 20380001
 253 03184853A2E1APICALL:DefWindowProc 9A 104-WM_SYSKEYDOWN 12 20380001
 254 031848540BD1APIRET:DefWindowProc 0
 255 021848543FC1MSGRET:0 0B3F0208 9A 104-WM_SYSKEYDOWN 12 20380001
 50 256 011848547D61APIRET:DispatchMessage 0
 257 01184854ACB1APICALL:GetMessage 0 0 0
 258 011848B8CC51APIRET:GetMessage TRUE (8A 104 73 203E0001 001C7127 (12E 7F))

```

359 011848B912B1APICALL:TranslateMessage (8A 104 73 203E0001 001C7127 (12E 7F) )
360 011848B9C911APIRET:TranslateMessage TRUE
361 011848BA22F1APICALL:DispatchMessage (8A 104 73 203E0001 001C7127 (12E 7F) )
362 021848BA9071MSGCALL:0B3F0208 9A 104-WM_SYSKEYDOWN 73 203E0001
363 031848BAFBF1APICALL:DefWindowProc 9A 104-WM_SYSKEYDOWN 73 203E0001
364 031848BB9301APIRET:DefWindowProc 0
365 021848BBC6C1MSGRET:0 0B3F0208 9A 104-WM_SYSKEYDOWN 73 203E0001
366 011848BC0651APIRET:DispatchMessage 0
367 011848BC35B1APICALL:GetMessage 0 0 0
368 011848BC9441APIRET:GetMessage TRUE (8A 112 F060 0 001C7136 (12E 7F) )
369 011848BCD491APICALL:TranslateMessage (8A 112 F060 0 001C7136 (12E 7F) )
370 011848BD2B21APIRET:TranslateMessage FALSE
371 011848BD5911APICALL:DispatchMessage (8A 112 F060 0 001C7136 (12E 7F) )
372 021848BEEC41MSGCALL:0B3F0208 8A 112-WM_SYSCOMMAND F060 0
373 031848BF5741APICALL:DefWindowProc 8A 112-WM_SYSCOMMAND F060 0
374 041848BFD1F1MSGCALL:0B3F0208 9A 10-WM_CLOSE 0 0
375 051848C00651APICALL:DefWindowProc 9A 10-WM_CLOSE 0 0
376 061848C086B1MSGCALL:0B3F0208 9A 46-WM_WINDOWPOSCHANGING 0 C370180
377 071848C0C201APICALL:DefWindowProc 9A 46-WM_WINDOWPOSCHANGING 0 C370180
378 071848C10DF1APIRET:DefWindowProc 0
379 061848C140E1MSGRET:0 0B3F0208 9A 46-WM_WINDOWPOSCHANGING 0 C370180
380 061848C2EEB1MSGCALL:0B3F0208 9A 47-WM_WINDOWPOSCHANGED 0 C370180
381 071848C32DC1APICALL:DefWindowProc 9A 47-WM_WINDOWPOSCHANGED 0 C370180
382 071848C37A11APIRET:DefWindowProc 0
383 061848C3E3B1MSGRET:0 0B3F0208 9A 47-WM_WINDOWPOSCHANGED 0 C370180
384 061848D755E1MSGCALL:0B3F0208 9A 86-WM_NCACTIVATE 0 0
385 071848D79611APICALL:DefWindowProc 9A 86-WM_NCACTIVATE 0 0
386 071848D80E31APIRET:DefWindowProc 1
387 061848D841D1MSGRET:1 0B3F0208 9A 86-WM_NCACTIVATE 0 0
388 061848D89AC1MSGCALL:0B3F0208 9A 6-WM_ACTIVATE 0 0 0
389 071848D8D131APICALL:DefWindowProc 9A 6-WM_ACTIVATE 0 0 0
390 071848D91621APIRET:DefWindowProc 0
391 061848D947B1MSGRET:0 0B3F0208 9A 6-WM_ACTIVATE 0 0 0
392 061848D9AC91MSGCALL:0B3F0208 9A 1C-WM_ACTIVATEAPP 0 0 0
393 071848D9E541APICALL:DefWindowProc 9A 1C-WM_ACTIVATEAPP 0 0 0
394 071848DA22F1APIRET:DefWindowProc 0
395 061848DA5331MSGRET:0 0B3F0208 9A 1C-WM_ACTIVATEAPP 0 0 0
396 061848DAB851MSGCALL:0B3F0208 8A 8-WM_KILLFOCUS 0 0
397 071848DAECE1APICALL:DefWindowProc 8A 8-WM_KILLFOCUS 0 0
398 071848DB26F1APIRET:DefWindowProc 0
399 061848DB5761MSGRET:0 0B3F0208 9A 8-WM_KILLFOCUS 0 0
400 061848DBB1E1MSGCALL:0B3F0208 8A 2-WM_DESTROY 0 0
401 071848DBE461APICALL:PostQuitMessage 0
402 071848DD8801APIRET:PostQuitMessage
403 061848DDDBB1MSGRET:0 0B3F0208 9A 2-WM_DESTROY 0 0
404 061848DE8021MSGCALL:0B3F0208 9A 92-WM_NCDESTROY 0 0
405 071848DEB6D1APICALL:DefWindowProc 8A 92-WM_NCDESTROY 0 0
406 071848DF6261APIRET:DefWindowProc 0
407 061848E0EE21MSGRET:0 0B3F0208 9A 92-WM_NCDESTROY 0 0
408 051848E46901APIRET:DefWindowProc 0
409 041848E49E81MSGRET:0 0B3F0208 9A 10-WM_CLOSE 0 0
410 031848E52371APIRET:DefWindowProc 0
411 021848E55821MSGRET:0 0B3F0208 9A 112-WM_SYSCOMMAND F060 0
412 011848E596B1APIRET:DispatchMessage 0
413 011848E5C611APICALL:GetMessage 0 0 0
414 011848E621C1APIRET:GetMessage FALSE (0 12 0 0 001C71CC (12E 7F) )

```

55 The Synthetic GUI Application

Once a log file for the execution of an application program is generated, the execution of the application program can be simulated on a new operating system. The synthetic GUI application (SGA) program reads

in the log file and simulates the recorded behavior by invoking the functions of the new operating system or some other existing operating system. By simulating the behavior of the application program, the new operating system can be tested to ensure its functions perform correctly.

The SGA program provides a thunk function for every real function of the old operating system. The
 5 thunk function simulates the execution of the real function on the new operating system by invoking the functions of the new operating system. If the new operating system has a function whose behavior corresponds to the behavior of the function of the old operating system, then the thunk function can use the function of the new operating system to simulate the behavior. However, if there is no corresponding function of the new operating system, then the thunk function may have to invoke several functions of the
 10 new operating system to simulate the behavior.

Each thunk function that corresponds to a real function, that is passed a callback routine, provides a thunk callback routine. The thunk callback routine simulates the behavior of the corresponding real callback routine in the application program. Each thunk callback routine, when invoked by the new operating system, invokes an SGAEngine routine, as described below, to simulate the behavior of the application program.

15 Figure 10A is an overview diagram illustrating the synthetic GUI application (SGA). The SGA 1010 comprises the SGAEngine routine 1050, thunk functions 1060 for each real function of the old operating system, and thunk callback routines 1070 corresponding to real callback routines of the new operating system. The new operating system 1020 comprises real functions 1030 and a window class table 1014. The SGA 1010 reads the log file and simulates the execution of the recorded application program. Each
 20 recorded invocation of a function is simulated by invoking the corresponding thunk function with the recorded passed parameter(s). The behavior of each invocation of a callback routine is simulated by a thunk callback routine.

The SGAEngine routine 1050 controls the simulation. The routine is passed a message and simulates the execution of that message. The routine scans the log file for the invocation of a callback routine through
 25 which the message is sent to the application. The routine then simulates the execution of each invocation of a function that is nested within that callback routine. For example, if the SGAEngine routine is invoked for the message WM_CREATE corresponding to line 20, then the routine simulates the nested invocation of the function DefWindowProc at line 21. The SGAEngine routine retrieves the passed parameters from the log file and invokes functions of the new operating system to simulate the behavior of the function of the old
 30 operating system. When the function of the new operating system is complete, the SGAEngine routine determines whether the returned parameters compare to the logged returned parameter. If the parameters do not compare, then the difference is noted. Optionally, timing information about the invocation on the new operating system can be recorded in a separate new log file.

Figure 10B is a flow diagram of the synthetic GUI application program. The synthetic GUI application
 35 (SGA) inputs a log file and simulates the execution of the GUI application program recorded in the log file by invoking functions of a new operating system. In step 1001, the SGA generates a sent message file (SMF). The sent message file contains an entry for each sent message recorded in the log file. Each entry in the sent message file contains an identifier of the message, the line number of the message in the log file, and a handle identifying the resource (e.g., window, dialog box) to which the message is directed. Table
 40 3 is a sample sent message file generated for the log file of Table 2. Line number 5 of the sent message file contains the entry "WM_CREATE 20, 8A", which indicates that line number 20 of the log file records that a WM_CREATE was sent to the window with handle 8A. In step 1002, the SGA generates a posted message file (PMF). The posted message file contains an entry for each posted message recorded in the log file. Each entry in the posted message file contains the message name, the line number corresponding
 45 to the retrieval of the message, the line number where the event corresponding to the posted message should be simulated, and a handle identifying the resource (e.g., window, dialog box) to which the message is directed. Table 4 contains a sample posted message file for the log file of Table 2. Line number 5 in the posted message file contains the entry "WM_KEYDOWN 192, 187, 8A", which indicates that line number 192 in the log file records at a WM_KEYDOWN message was retrieved by the window procedure for the
 50 window with handle 8A and that the posting of the message should be simulated at line 187. Alternatively, a sent message file and a posted message file is generated by the logger during the execution of the application program. In step 1003, the SGA invokes a SGAEngine routine to simulate execution of the application program.

TABLE 3
SENT MESSAGE FILE

5	1	INIT	0,8A
	2	WM_GETMINMAXINFO	8,8A
	3	WM_NCCREATE	12,8A
	4	WM_NCCALCSIZE	16,8A
	5	WM_CREATE	20,8A
	6	WM_SHOWWINDOW	26,8A
10	7	WM_WINDOWPOSCHANGING	30,8A
	8	WM_QUERYNEWPALETTE	34,8A
	9	WM_WINDOWPOSCHANGING	38,8A
	10	WM_ACTIVATEAPP	40,8A
	11	WM_NCACTIVATE	46,8A
15	12	WM_GETTEXT	48,8A
	13	WM_ACTIVATE	54,8A
	14	WM_SETFOCUS	56,8A
	15	WM_NCPAINT	67,8A
	16	WM_GETTEXT	64,8A
20	17	WM_ERASEBKGD	70,8A
	18	WM_WINDOWPOSCHANGED	74,8A
	19	WM_SIZE	78,8A
	20	WM_MOVE	82,8A
	21	WM_PAINT	88,8A
	22	WM_NCHITTEST	126,8A
25	23	WM_SETCURSOR	130,8A
	24	WM_MOUSEMOVE	138,8A
	25	WM_NCHITTEST	144,8A
	26	WM_SETCURSOR	148,8A
	27	WM_MOUSEMOVE	156,8A
30	28	WM_NCHITTEST	162,8A
	29	WM_SETCURSOR	166,8A
	30	WM_MOUSEMOVE	174,8A
	31	WM_KEYDOWN	184,8A
	32	WM_KEYDOWN	194,8A
	33	WM_CHAR	204,8A
35	34	WM_KEYUP	214,8A
	35	WM_KEYUP	224,8A
	36	WM_KEYDOWN	234,8A
	37	WM_CHAR	244,8A
	38	WM_KEYUP	254,8A
40	39	WM_KEYDOWN	264,8A
	40	WM_CHAR	274,8A
	41	WM_KEYUP	284,8A
	42	WM_KEYDOWN	294,8A
	43	WM_CHAR	304,8A
	44	WM_KEYUP	314,8A
45	45	WM_KEYDOWN	324,8A
	46	WM_CHAR	334,8A
	47	WM_KEYUP	344,8A
	48	WM_SYSKEYDOWN	354,8A
	49	WM_SYSKEYDOWN	364,8A
50	50	WM_SYSCOMMAND	374,8A
	51	WM_CLOSE	376,8A
	52	WM_WINDOWPOSCHANGING	378,8A

53	WM_WINDOWPOSCHANGED	382,8A
54	WM_NCACTIVATE	386,8A
55	WM_ACTIVATE	390,8A
56	WM_ACTIVATEAPP	394,8A
57	WM_KILLFOCUS	398,8A
58	WM_DESTROY	402,8A
59	WM_NCDESTROY	406,8A

TABLE 4

POSTED MESSAGE FILE1		
1	WM_MOUSEMOVE	136,123,8A
2	WM_MOUSEMOVE	154,141,8A
3	WM_MOUSEMOVE	172,159,8A
4	WM_KEYDOWN	182,177,8A
5	WM_KEYDOWN	192,187,8A
6	WM_CHAR	202,197,8A
7	WM_KEYUP	212,207,8A
8	WM_KEYUP	222,217,8A
9	WM_KEYDOWN	232,227,8A
10	WM_CHAR	242,237,8A
11	WM_KEYUP	252,247,8A
12	WM_KEYDOWN	262,257,8A
13	WM_CHAR	272,267,8A
14	WM_KEYUP	282,277,8A
15	WM_KEYDOWN	292,287,8A
16	WM_CHAR	302,297,8A
17	WM_KEYUP	312,307,8A
18	WM_KEYDOWN	322,317,8A
19	WM_CHAR	332,327,8A
20	WM_KEYUP	342,337,8A
21	WM_SYSKEYDOWN	352,347,8A
22	WM_SYSKEYDOWN	362,357,8A
23	WM_SYSCOMMAND	372,367,8A

Figure 11 is a flow diagram of the routine GenerateSentMessageFile. The sent message file is generated to improve the processing efficiency of the SGAEngine routine. In step 1101, the routine writes an initialization message (INIT) to the sent message file. This initialization message is used as an indication to initialize the SGAEngine routine. In steps 1102 through 1105, the routine loops searching for invocations of a callback routine and writes entries to the sent message file. In step 1102, the routine searches the log file for the next entry corresponding to the invocation of a callback routine. In step 1103, if the end of the log file is reached, then the routine returns, else the routine continues at step 1104. In step 1104, the routine writes the message, the line number of the entry, and the handle of the resource to which the message is directed to the sent message file and loops to step 1101.

Figure 12 is a flow diagram of the routine GeneratePostedMessageFile. This routine searches the log file for each posted message and determines when the posted message should be simulated by the SGA. In step 1201, the routine searches for the next invocation of the function GetMessage, which retrieves posted messages in the log file, or function PeekMessage, which looks at posted messages and may or may not retrieve them. In step 1202, if the end of the log file is reached, then the routine returns, else the routine continues at step 1203. In step 1203, the routine records the line number in the log file corresponding to the invocation of the function GetMessage or PeekMessage. This line number indicates the point at which the event corresponding to the posted message should be simulated. For example, if the posted message is WM_KEYDOWN, then the WM_KEYDOWN message should be simulated before the SGA simulates the execution of the function GetMessage that retrieves the WM_KEYDOWN message or before the execution of the function PeekMessage, which may or may not have noticed its presence in the

message queue. In step 1204, the routine searches the log file for an entry corresponding to the invocation of a callback routine that matches the posted message. That is, the invocation of the callback routine through which the posted message was sent to the application program. In step 1205, the routine records the line number of the entry in the log file corresponding to the invocation of the callback routine. In step 1206, the routine writes the message, message line number, post line number, and the handle of the resource to which the message is directed to the posted message file and loops to step 1201. In an alternate embodiment, the sent message and posted message files are generated during a single pass of the log file. Also, the sent message and posted message files need not be generated, rather the information can be determined from the log file as the simulation proceeds.

Figure 13 is a flow diagram of the SGAEngine routine. This routine is passed a window handle and a message. The routine simulates the application program processing of that message as recorded in the log file. The routine determines which entry in the log file corresponds to the passed message. In a preferred embodiment, this determination selects the first entry in the log file of a message of the same type as the passed message and with the same handle to a resource. The routine then simulates the execution of the application program that occurs in response to the receipt of the message. The execution that occurs in response is represented by the entries in the log file that are nested within the invocation of the callback routine that processes the message. The SGAEngine routine is called recursively to process nested messages. To initiate the SGAEngine routine, a dummy entry is stored at line number 0 in the log file at nesting level 0 and corresponding to the sending of the message INIT. The SGAEngine routine simulates execution of all entries in the log file nested within nesting level 0, which is the entire log file. In step 1301, the routine finds the message in the sent message file that corresponds to the passed message and records that message line. In step 1302, the routine determines the nesting level of the found message. In step 1303, the routine validates the passed parameters. In step 1304, the routine searches for the next line in the log file that corresponds to an invocation of a function within the callback routine that processed the passed message. The routine records that line number as the call line. For example, if the passed message corresponds to the message at line number 8 of the log file, then line number 9 contains a nested function invocation. In step 1305, if the routine finds a next line, then the routine continues at step 1306, else the routine continues at step 1310. In step 1306, the routine invokes routine SimulatePostedMessage to simulate the posting of a message, if appropriate. In step 1307, the routine finds the line corresponding to the return of the nested function invocation. In step 1308, the routine retrieves or identifies the thunk function for the nested function. In step 1309, the routine invokes the thunk function and loops to step 1304. In step 1310, the routine finds the return line associated with the invocation line of the message found in step 1301. In step 1311, the routine sets the appropriate return values and returns.

Figure 14 is a flow diagram of the routine FindMessageInSentMessageFile. This routine is passed a message and a resource (window) handle. This routine finds the next message in the sent message file that corresponds to the passed message for the passed resource handle. The routine returns the line number in the log file of the message. The routine maintains an array to track the last entry in the sent message file found for each message type for each resource handle. The routine starts the search from the last entry. In step 1401, the routine retrieves the message line of the last message found in the sent message file for the passed resource handle of the passed message type. In steps 1402 through 1403, the routine loops searching the sent message file for the next message of the passed message type for the passed resource handle. In step 1402, the routine increments the message line. In step 1403, if the message of the message line in the sent message file equals the passed message type and passed resource handle, then the routine continues at step 1404, else the routine loops to step 1402. In step 1404, the routine sets the last message line number for the passed message type and passed resource handle to the message line. In step 1405, the routine retrieves the line number in the log file of the entry corresponding to the invocation of the callback routine to process the passed message and returns.

Figure 15 is a flow diagram of the routine SimulatePostedMessage. This routine scans the posted message file to determine if the posting of a message should be simulated before the SGA simulates the execution of the function corresponding to the entry in the log file at the passed line number. The routine then simulates the posting of the message, if appropriate. In step 1501, the routine scans the posted message file to determine whether a message should be posted before simulation of the entry at the passed line number in the log file. In step 1502, if a message should be posted, the routine continues at step 1503, else the routine returns. In step 1503 through 1510, the routine determines which message should be posted and calls the appropriate routine to simulate the posting of the message and the routine then returns.

Figure 16 is a flow diagram of a routine that simulates the posting of the WM_KEYDOWN message. The routine is passed the sent line corresponding to the invocation of the callback routine and the post line

number corresponding to the line number before which the message should be posted. In step 1601, the routine retrieves the message at sent line from the log file. In step 1602, the routine calls the function PostMessage or the new operating system or its equivalent to post the retrieved WM_KEYDOWN message and the routine returns.

5 Figure 17 is a flow diagram of a routine that simulates a WM_MOUSEMOVE message. The routine is passed the sent line number in the log file of the entry corresponding to the invocation of the callback routine and the post line number corresponding to the line number before which the posting of the message should be simulated. To simulate a WM_MOUSEMOVE message, the routine simulates the sent messages that were sent during the invocation of the function GetMessage that retrieved the posted
10 WM_MOUSEMOVE message. The routine then posts the WM_MOUSEMOVE message to the new operating system. For example, line number 136 of the log file corresponds to the sending of a WM_MOUSEMOVE message and line number 123 corresponds to the line before which the posting of the WM_MOUSEMOVE message should be simulated. To simulate the WM_MOUSEMOVE message, the sending of the WM_NCHITTEST message at line 124 and the sending of the WM_SETCURSOR message at line 128 should be simulated. This routine uses the function SendMessage (or its equivalent) in the new
15 operating system to send these messages. The thunk callback routines are then invoked by the new operating system to send the message to the SGA. After the messages are sent, then the SGA posts the WM_MOUSEMOVE message by invoking the function PostMessage (or its equivalent) in the new operating system. In steps 1701 through 1704, the routine retrieves each sent message and sends the message to the new operating system. In step 1701, the routine sets the start of the search at the post line. In step 1702, if the search line in the log file indicates the return from the invocation of the function GetMessage at the post line, then the routine continues at step 1705, else the routine continues at step 1703. In step 1703, the routine retrieves the next sent message from the log file. In step 1704, the routine calls the function SendMessage of the new operating system with the retrieved message and loops to step
20 1702. In step 1705, the routine retrieves the message corresponding to the return from the invocation of the function GetMessage at the post line. In step 1706, the routine calls the function PostMessage of the new operating system with the retrieved message and returns. Alternatively, the routine that simulates a WM_MOUSEMOVE message could invoke a function SetCursorPos of the new operating system, which effectively simulates a WM_MOUSEMOVE message.

30 Figure 18 is a flow diagram of a thunk window procedure. Each thunk callback routine operates in a similar way. In step 1801, the routine determines which old window handle corresponds to the window handle passed by the new operating system. The new operating system uses handles that do not necessarily correspond to the actual handles used by the old operating system. Consequently, the SGA maintains a mapping between handles of the old operating system and corresponding handles in the new
35 operating system. In step 1802, the routine invokes the SGAEngine routine to simulate the processing of the window procedure by the application program and returns.

Figure 19 is a flow diagram of the routine ThunkRegisterClass. In step 1901, the routine selects the next unused thunk window procedure. In step 1902, the routine establishes a correspondence between the class name and the selected thunk window procedure. Every time a window is created of this class, the selected
40 window procedure is used as the thunk window procedure based on this correspondence. In step 1903, the routine retrieves the parameters for the invocation of the function RegisterClass from the log file at the line number indicated by the passed call line. In step 1904, the routine sets the address of the window procedure to that of the selected thunk window procedure. In step 1905, the procedure calls the function RegisterClass of the new operating system. In step 1906, the routine determines if the return values of the invocation of function RegisterClass for the new operating system corresponds to the return values of the
45 invocation of the function RegisterClass for the old operating system as indicated by the return line in the log file and reports an error message if they do not correspond. The routine then returns.

Figure 20 is a flow diagram of the routine ThunkCreateWindow. In step 2001, the routine retrieves the passed parameters for the invocation of function CreateWindow from the log file at the line number
50 indicated by the passed call line. In step 2002, the routine determines the new window handle for the passed parent window, if one has been assigned. In step 2003, the routine determines the new menu handle of the passed menu, if one has been assigned. In step 2004, the routine determines the new handle for the instance of this application, the SGA. In step 2005, the routine gets the return parameters from the log file at the line number indicated by the passed return line. In step 2006, the routine calls the function
55 CreateWindow of the new operating system, passing the new parameters (e.g., new handles). In step 2007, the routine establishes the correspondence between the new handle returned by the new operating system and the old handle used by the old operating system as indicated by the return parameters in the log file. In step 2008, the routine determines whether the return parameters from the invocation of the function

CreateWindow of the new operating system corresponds to the return parameters of the invocation of function CreateWindow in the old operating system as indicated by the return line in the log file and reports an error if they do not correspond. The routine then returns.

Figure 21 is a flow diagram of the procedure ThunkDestroyWindow. In step 2101, the routine retrieves the passed parameters from the log file at the line number indicated by the passed call line. In step 2102, the routine determines the new window handle that corresponds to the old window handle. In step 2103, the routine retrieves the returned parameters from the log file at the line number indicated by the passed return line. In step 2104, the routine invokes the function DestroyWindow of the new operating system. In step 2104, the routine compares the return value of the invocation of the function DestroyWindow of the new operating system with the invocation of the function DestroyWindow of the old operating system as indicated by the retrieved return parameters and reports an error if there is a difference. The routine then returns.

Figure 22 is a flow diagram of a template thunk function. The thunk function is passed the line number in the log file corresponding to an invocation (call line) and return (return line) of the function. In steps 2201 through 2202, the thunk function retrieves the passed parameters from the log file. In steps 2203 through 2204, the thunk function retrieves the returned parameters from the log file. In step 2205, the thunk function retrieves the new handles that correspond to any old handle passed as a parameter. In step 2206, the routine invokes functions to simulate the function in the old operating system in the new operating system. In step 2207, the thunk function sets the correspondence between any new handles returned by the new operating system and the old handles of the old operating system. In step 2208, if the returned parameters from the simulated function do not correspond to the return parameters from the log file, then an error is reported in step 2209 and then the routine returns.

Figure 23 is a block diagram of an alternate embodiment of the present invention. A real-time logger 2302 intercepts service requests (function calls) of an application program 2301 intended for an old server program and sends the request to a simulator program 2303. The simulator program maps the received requests to service requests of the new server program 2304. The new server program performs these requests and passes results through the simulator to the real-time logger, which returns to the application program. Similarly, when the new server program invokes a callback routine, the simulator program routes the invocation to the logger which invokes the appropriate callback routine of the application program. The new server program inputs external events and routes indicators of events to the application program through the simulator and real-time logger.

One skilled in the art would appreciate that methods and systems of the present invention can be used to simulate the execution of programs that use an API provided by other than an operating system in conjunction with a graphical user interface. For example, a database system (server program) may provide various functions that comprise its API. The interaction between application programs (client program) and the database system can be recorded during an execution of the application program. This recorded interaction can then be used to simulate the application program and to test a new database system.

One skilled in the art would also appreciate that the new operating system could be simply the old operating system with certain functions rewritten, enhanced, or modified. In this case, although existing application programs can execute under the new operating system, the simulation system can be used to check the parameters returned by functions of the new operating system and parameters passed to callback routines by the new operating system.

One skilled in the art would further appreciate that the present invention can be used to compare the performance of a client program developed to request services of a first server program with simulated performance on a second server program. To compare performances, the client program is executed and a log is generated. The execution is simulated on both the first and second server programs and performance characteristics are recorded. The performance characteristics include simulation overhead for both server programs to provide a normalized basis for comparison. Alternatively, a similar approach can be used to compare the performance with a third server program.

Although the present has been described in terms of a preferred embodiment, it is not intended that the invention be limited to this embodiment. Modifications within the spirit of the invention will be apparent to those skilled in the art. The scope of the present invention is defined by the claims which follow.

Claims

1. A method in a computer system for simulating an execution of a client program, the client program for requesting services of a first server program, the client program specifying parameters for identifying variations in behavior of requested services, the method comprising the steps of:

during an execution of the client program, logging a plurality of requests for services of the first server program, each request including an identification of the requested service and any parameters specified with the requested service; and

5 requesting a second server program to perform a behavior similar to the behavior performed by the first server program based on the logged requests whereby the behavior of the execution of the client computer program is simulated.

2. The method of claim 1 wherein services are requested by invoking functions provided by the server programs and wherein the step of requesting the second server program to perform a behavior
10 includes the step of invoking functions provided by the second server program.

3. The method of claim 1 wherein the step of requesting the second server program to perform a behavior includes the steps of:
for each of a plurality of logged requests,
15 selecting one or more services of the second server program to simulate the behavior of the logged request; and
requesting the second server program to perform the selected services.

4. The method of claim 3 wherein the step of requesting the second server program to perform the selected services includes the step of passing to the second server program any parameters specified
20 with the logged request.

5. The method of claim 4 including the step of before passing a parameter to the second server program, mapping the parameter from a parameter generated by the first server program to a parameter
25 generated by the second server program.

6. The method of claim 5 wherein the step of mapping includes the step of mapping a computer system generated handle.

30 7. The method of claim 1 wherein services are requested by invoking functions provided by the server programs and wherein the step of requesting the second server program to perform a behavior includes the steps of:
associating a thunk function with each function provided by the first server program, each thunk function having a prototype similar to the prototype of the associated function of the first server
35 program; and
invoking the thunk function associated with the function of a logged request to simulate the behavior of logged request.

8. The method of claim 7 wherein the step of invoking the thunk function includes the step of passing a
40 parameter analogous to a parameter of the logged request.

9. The method of claim 8 including the step of before passing a parameter to the second server program, mapping the parameter from a parameter generated by the first server program to a parameter
45 generated by the second server program.

10. The method of claim 7 including the step of:
during the invocation of the thunk function, invoking one or more functions of the second server
program to simulate the behavior of the function associated with the thunk function.

50 11. The method of claim 10 including the step of:
upon return from a function of the second server program, specifying return parameters to be returned by the thunk function.

12. The method of claim 1 wherein services are requested by invoking functions provided by the server
55 programs and wherein the step of logging a plurality of requests for a service of the first server program includes the step of logging each return from an invoked function, each logged return including an identification of the requested service and any parameter returned by the invoked function.

13. The method of claim 1 wherein the step of logging requests includes the step of sending the request to a simulation program that controls the requesting of the second server program.
14. The method of claim 13 including the step of suppressing the sending of requests to the first server program.
15. The method of claim 1 wherein the step of logging includes the step of logging invocations of callback routines, and including the step of associating a thunk callback routine with each callback routine wherein the thunk callback routine simulates the behavior of the associated callback when invoked by the second server program.
16. The method of claim 1 wherein the first server program is an operating system and the second server program is a different version of the same operating system.
17. The method of claim 1 wherein the first server program is an operating system and the second server program is a different operating system.
18. A method in a computer system for comparing the performance between execution of a client program with a first server program and execution of the client program with a second server program, the client program for requesting services of a third server program, the method comprising the steps of:
 - during an execution of the client program, logging a plurality of requests for a service of the third server program, each request including an identification of the requested service;
 - simulating the execution of the client program with the first server program by selecting logged requests, requesting the first server program to perform the requested behavior of each selected logged request, and recording characteristics of the performance of the request by the first server program;
 - simulating the execution of the client program with the second server program by selecting logged requests, requesting the second server program to perform the requested behavior of each selected logged request, and recording characteristics of the performance of the request by the second server program; and
 - analyzing the recorded characteristics of the performance of requests by the first server program and the second server program.
19. The method of claim 18 wherein the steps of recording the performance characteristics record timing information.
20. The method of claim 19 wherein the step of simulating the execution of the client program with the first server program are performed on a first computer and the step of simulating the execution of the client program with the second server program is performed on a second computer, the first computer and the second computer having different architectures.
21. The method of claim 20 wherein the step of logging is performed on a third computer having an architecture different from the architecture of the first computer and the second computer.
22. A method in a computer system for cooperatively executing a client computer program with a second server program, the client computer program developed for requesting services of a first server program, the client program specifying a service name and parameters for the requested services, the method comprising the steps of:
 - during an execution of the client program on the computer system,
 - intercepting a plurality of requests for services of the first server program, each request including a service name and parameters; and
 - sending each intercepted request for service to a mapping program;
 - upon receipt of a sent request by the mapping program, requesting the second service program to perform the behavior of the received request; and
 - sending to the client program any parameters returned by the second service program whereby the client program continues execution.

23. The method of claim 22 wherein the client program executes on a first computer and the second server program executes on a second computer.
24. The method of claim 23 wherein the mapping program executes on the same computer as the client program and including the step of sending requests for services of the second service program from the first computer to the second computer.
25. The method of claim 22 wherein the services are requested by invoking functions provided by the service programs and wherein the step of requesting the second server program to perform a behavior includes the step of invoking functions provided by the second server program.
26. The method of claim 22 wherein the step of requesting the second server program to perform a behavior includes the steps of:
- for a plurality of sent requests,
 - selecting one or more services of the second server program to simulate the behavior of the sent requests; and
 - requesting the second server program to perform the selected services.

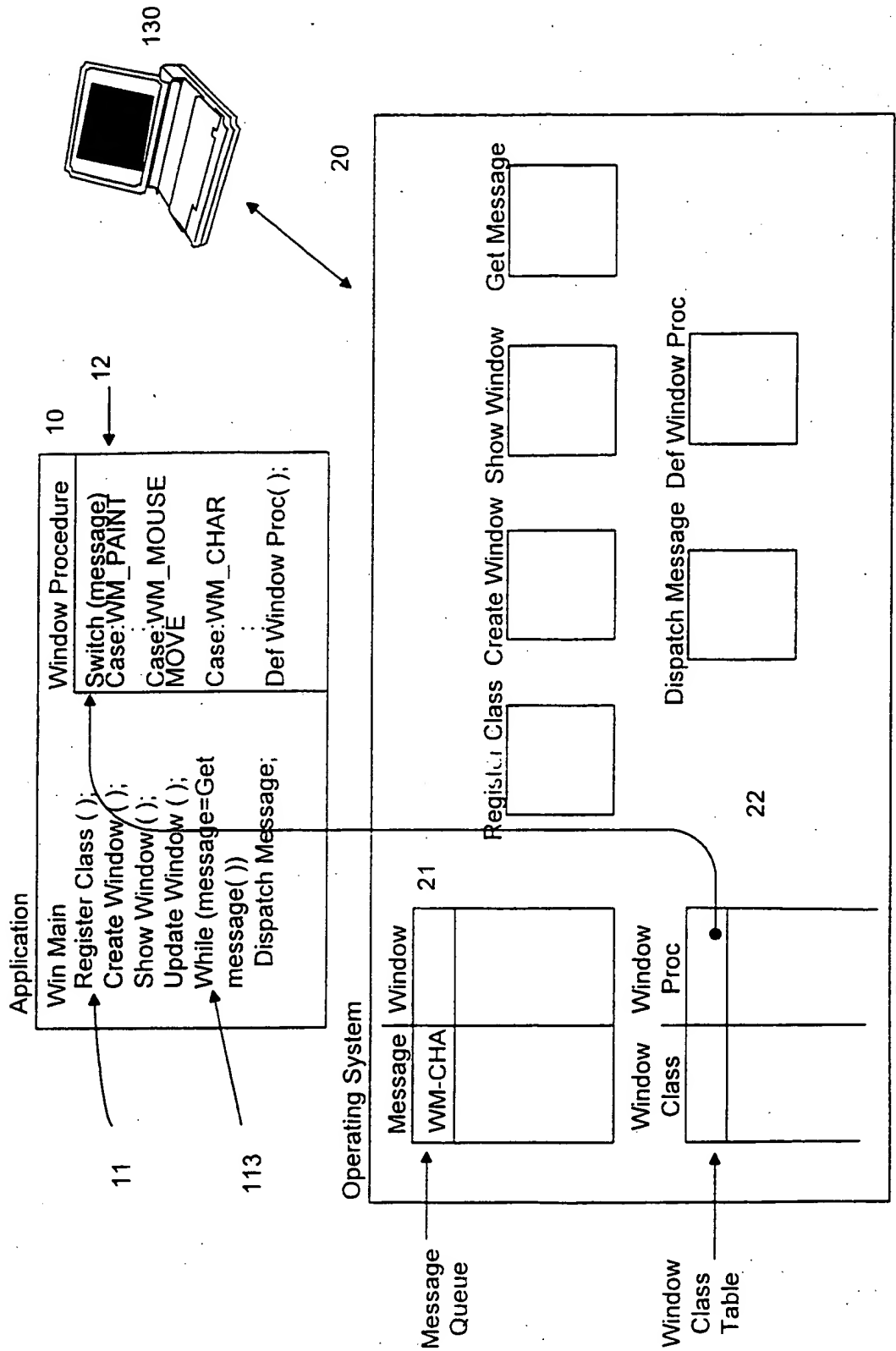


FIG. 1

Prior Art

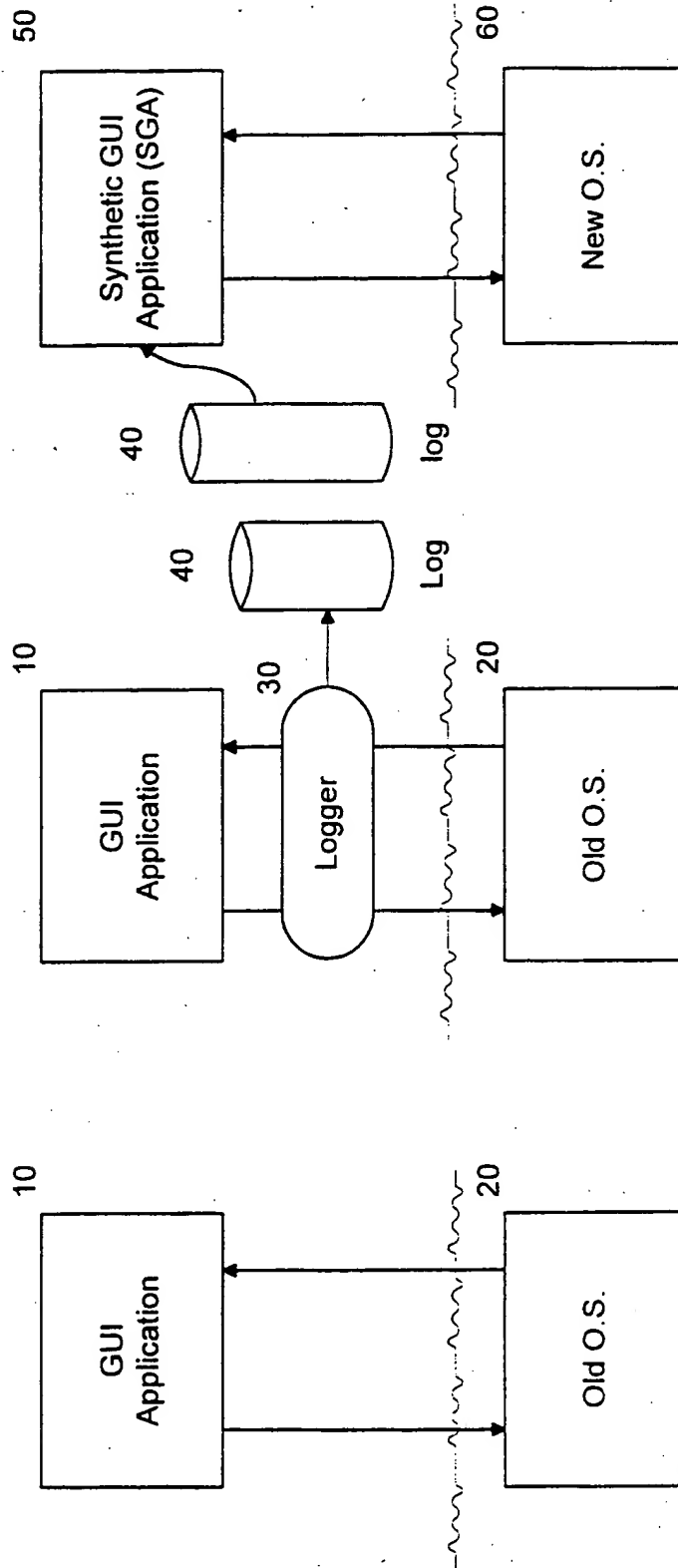


FIG. 2A

FIG. 2B

FIG. 2C

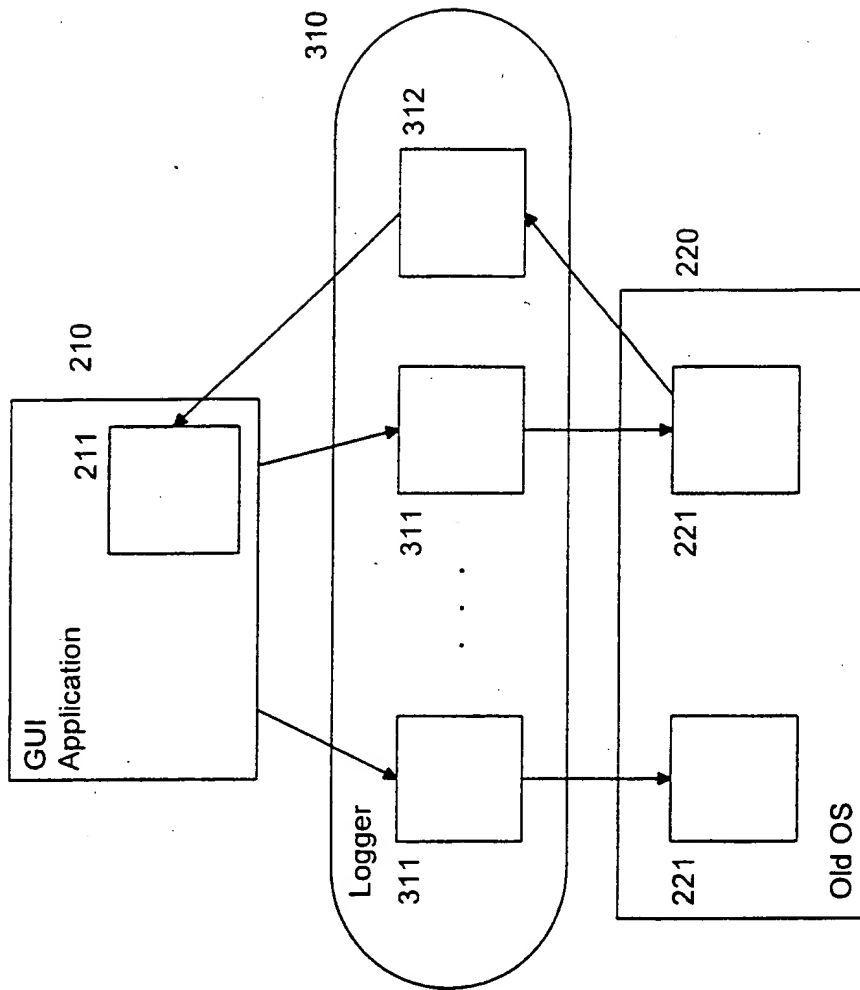


FIG. 3

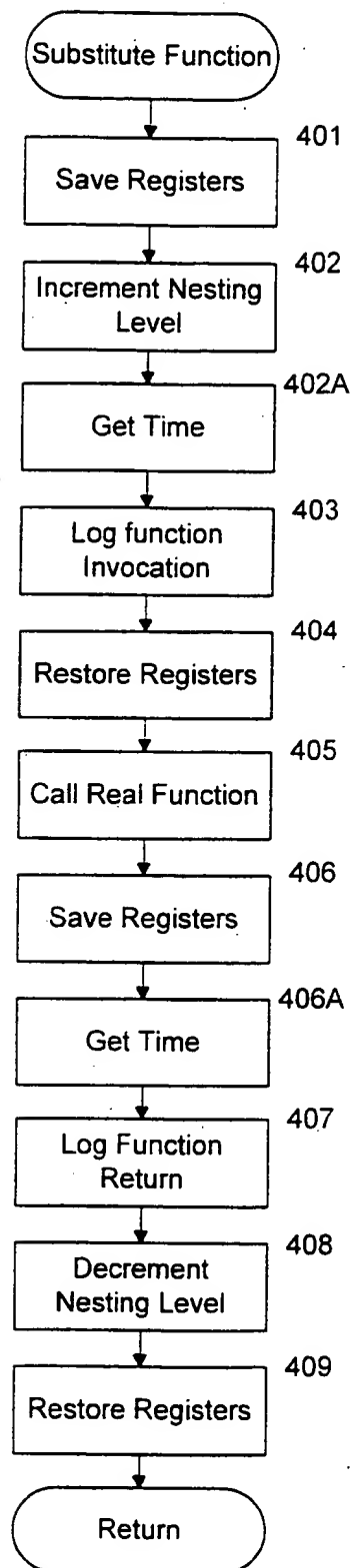
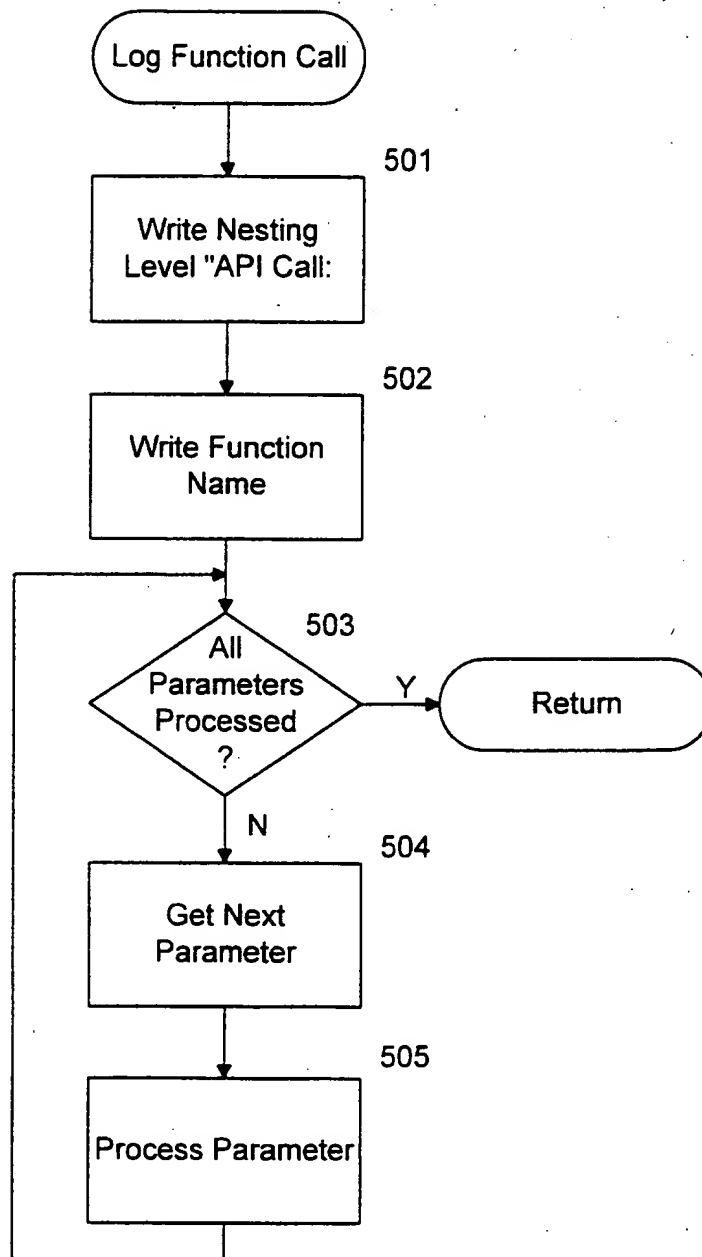


FIG. 4

**FIG. 5**

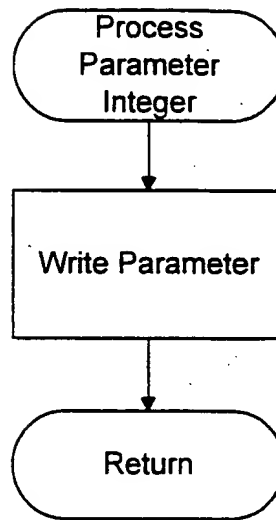
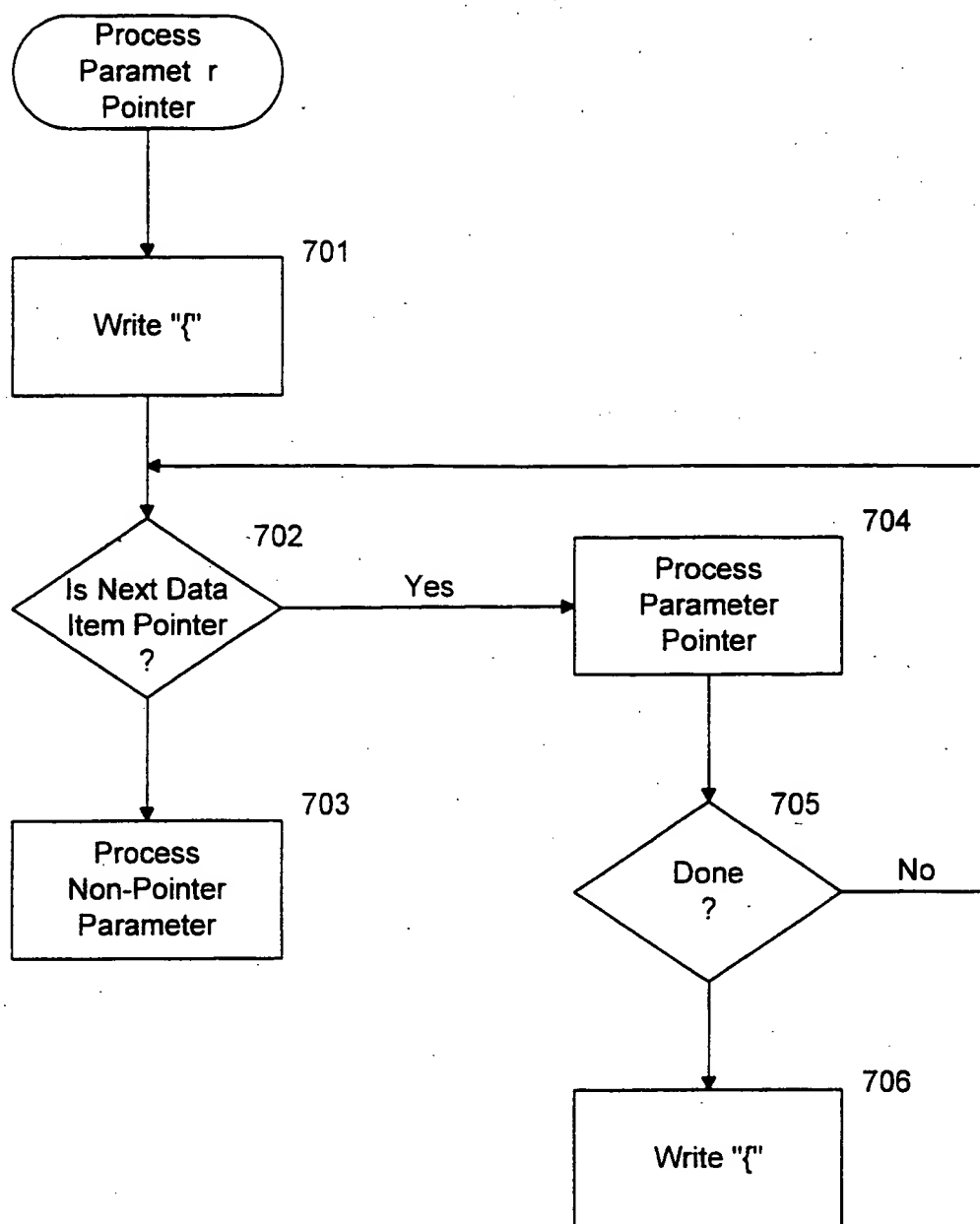


FIG. 6

**FIG. 7**

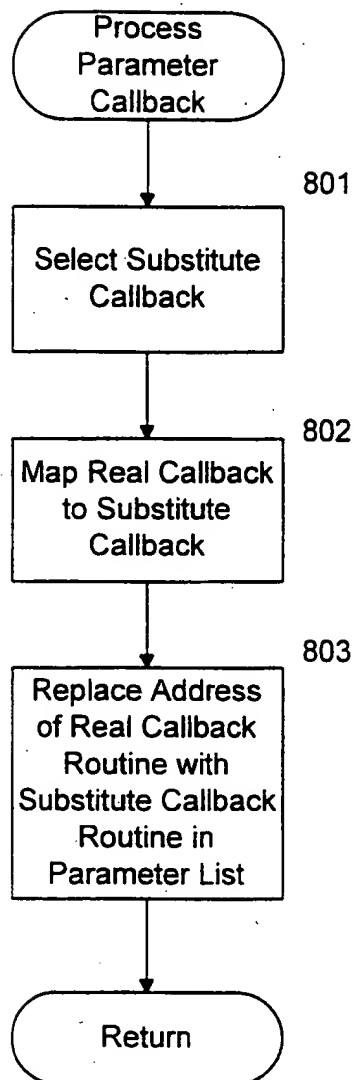
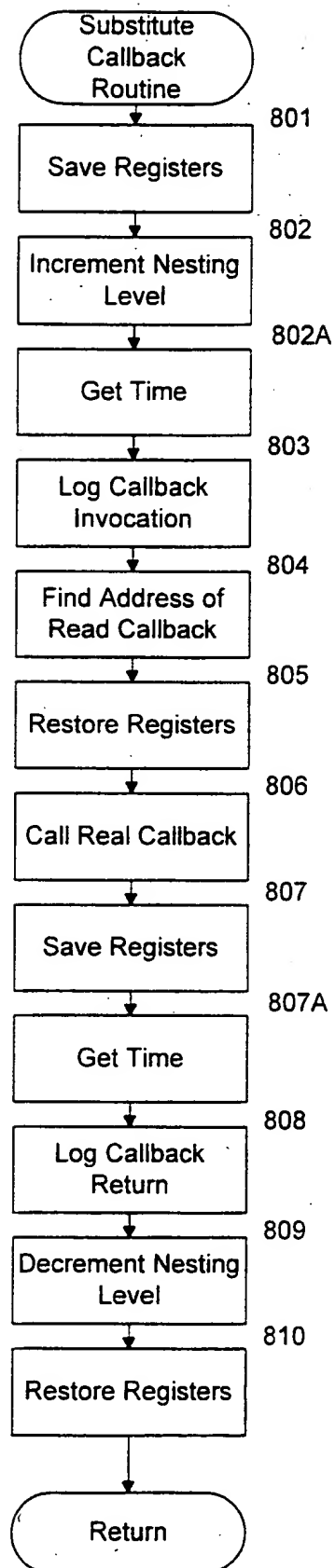


FIG. 8

**FIG. 9**

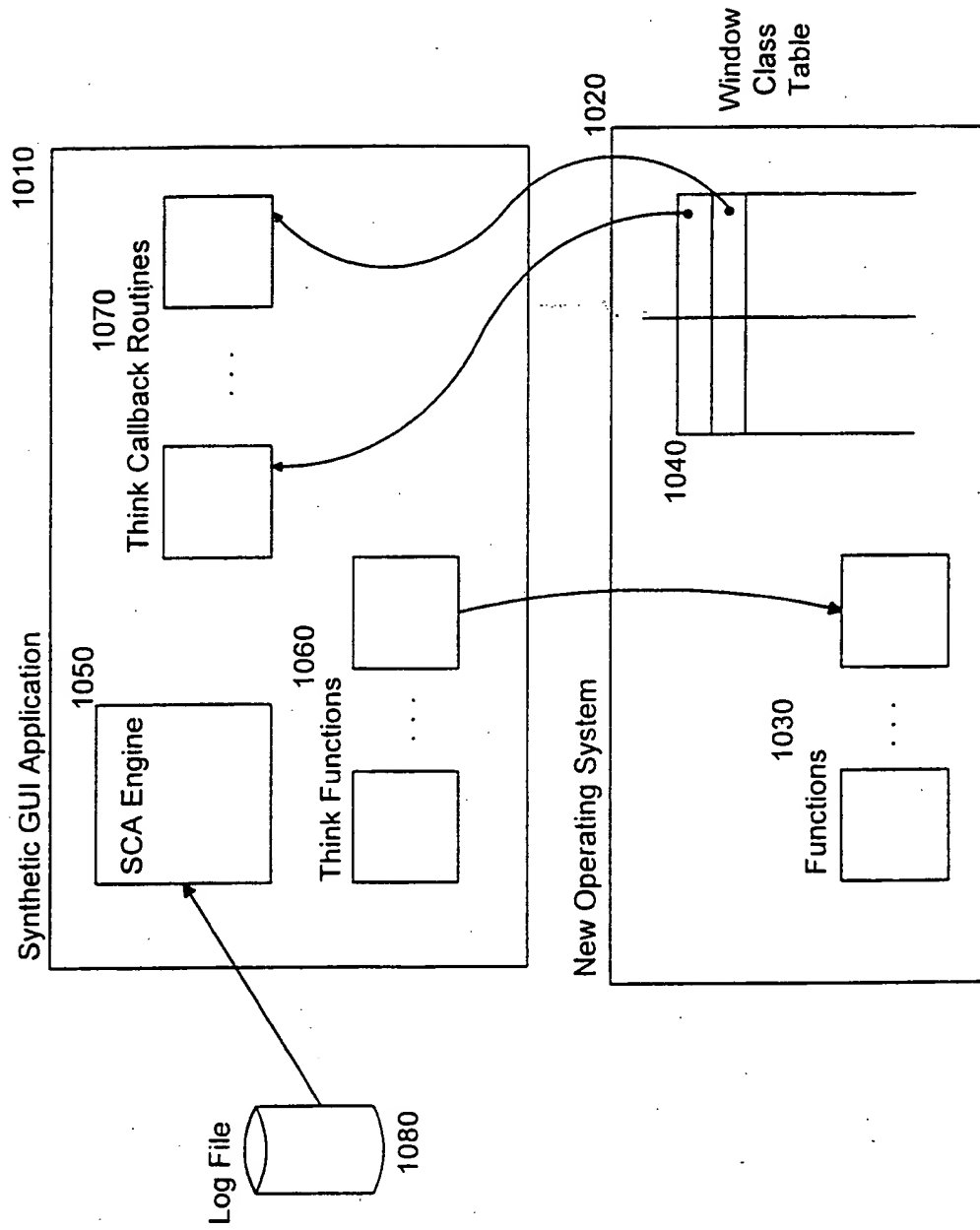


FIG. 10A

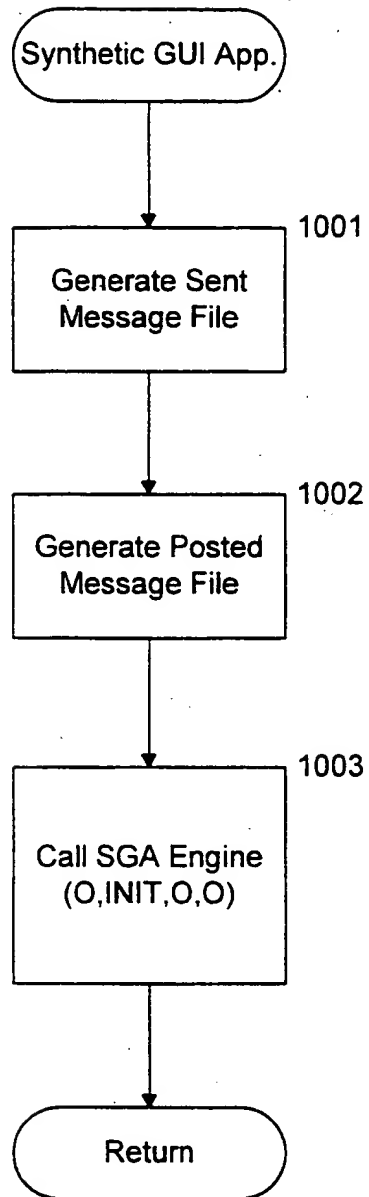
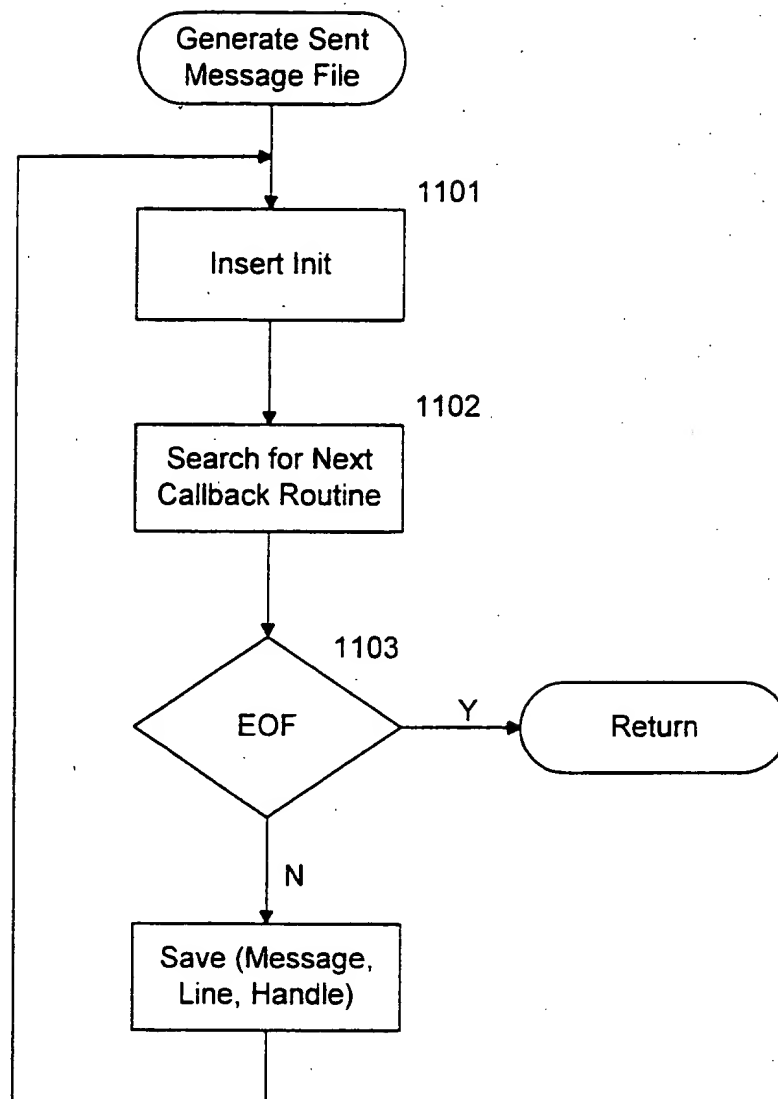
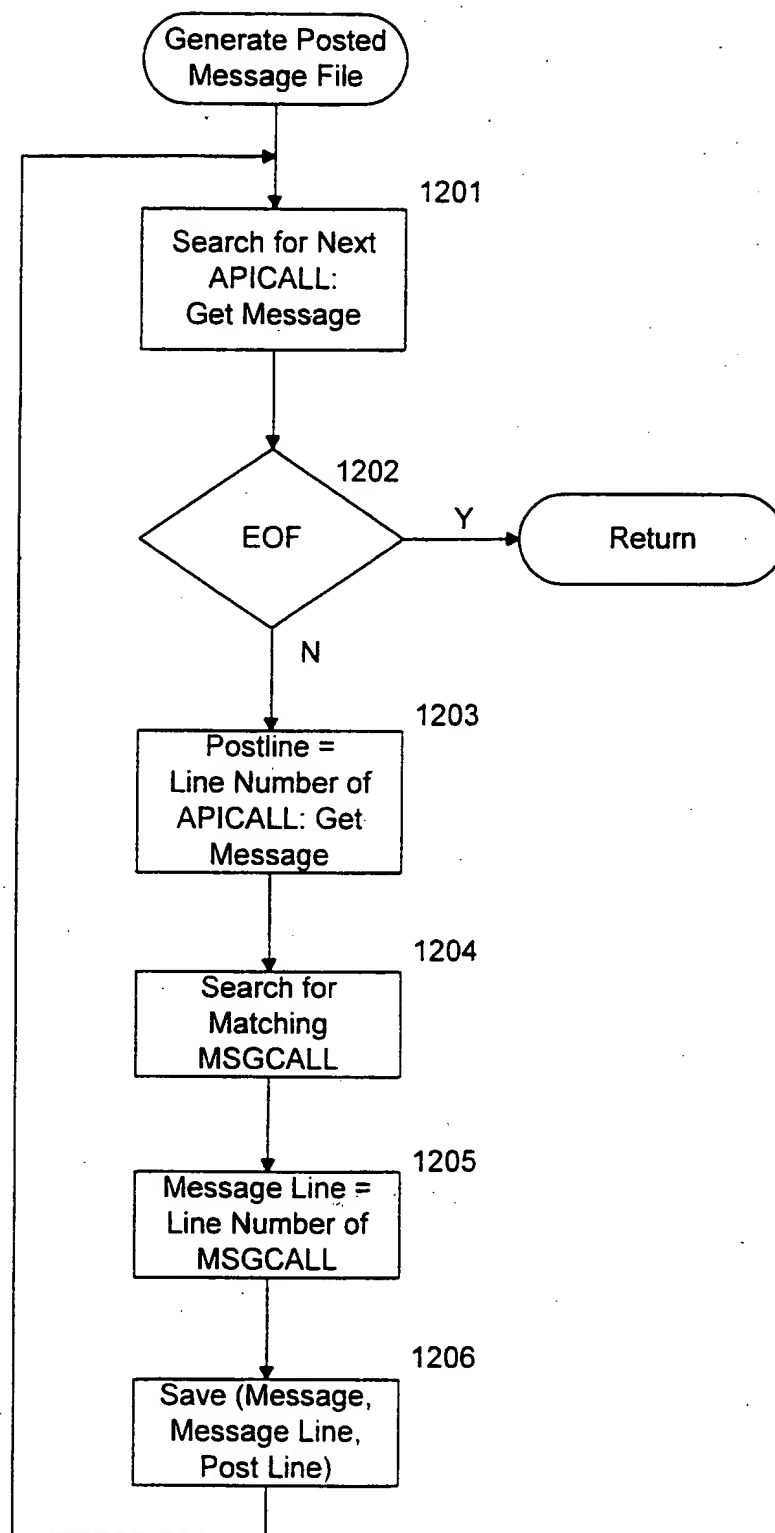


FIG. 10B

**FIG. 11**

**FIG. 12**

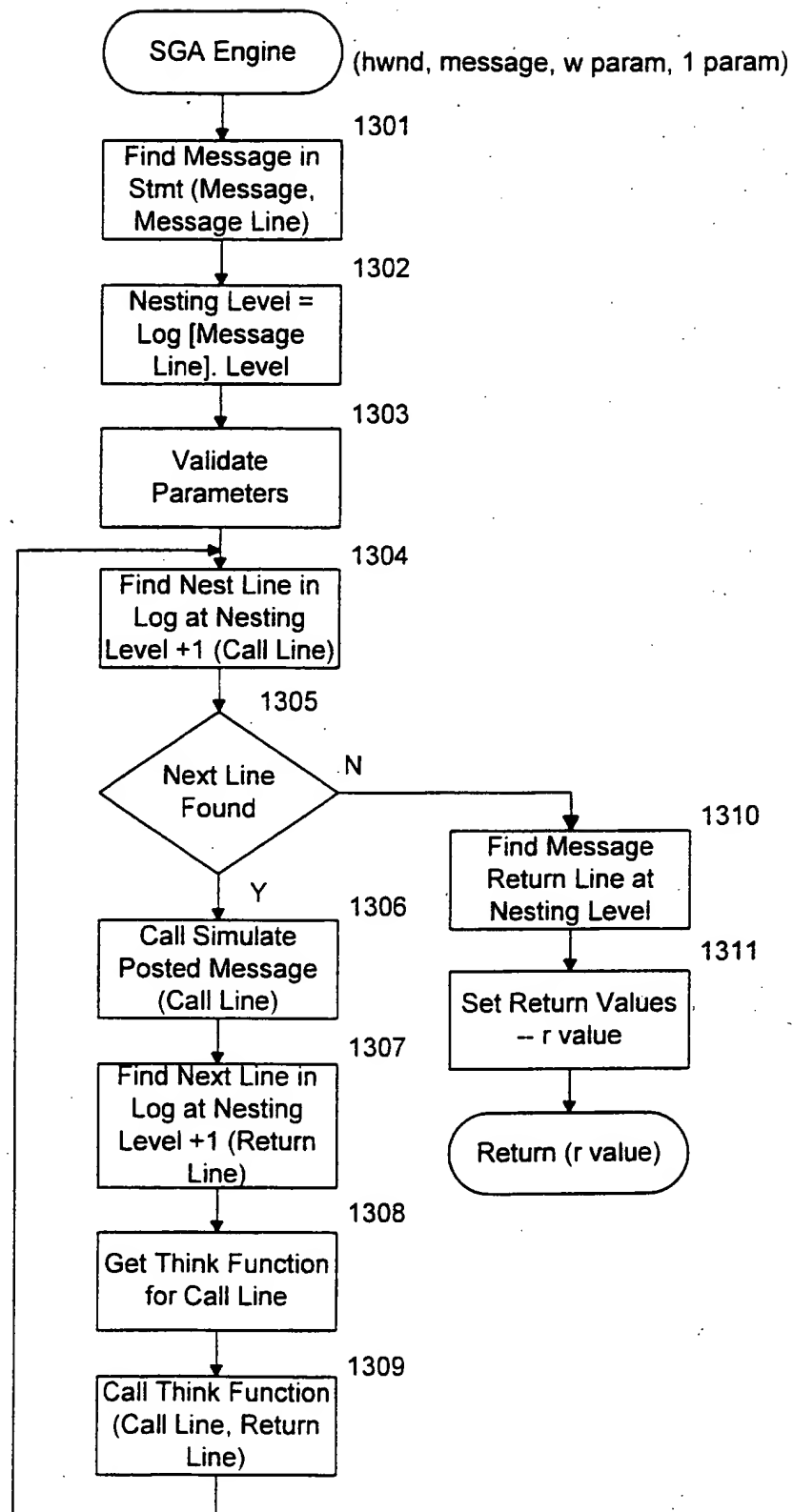
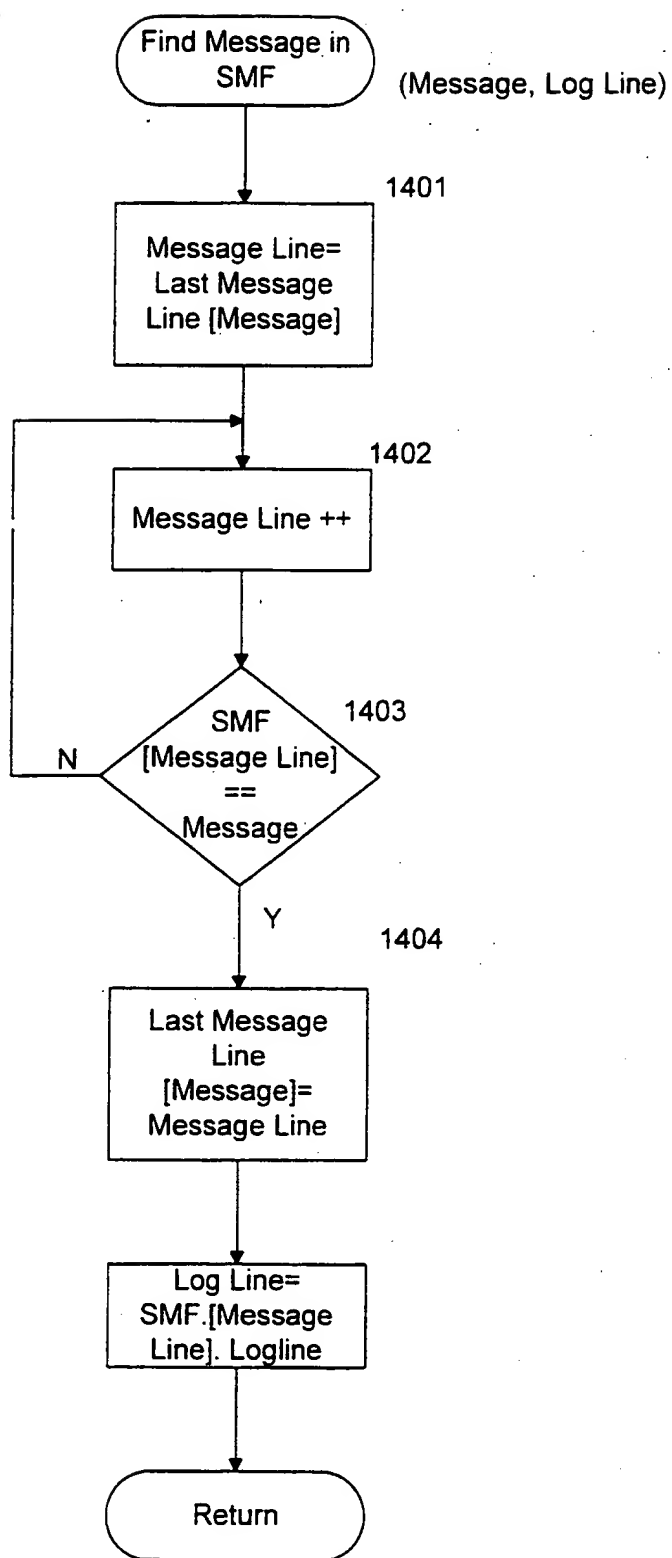
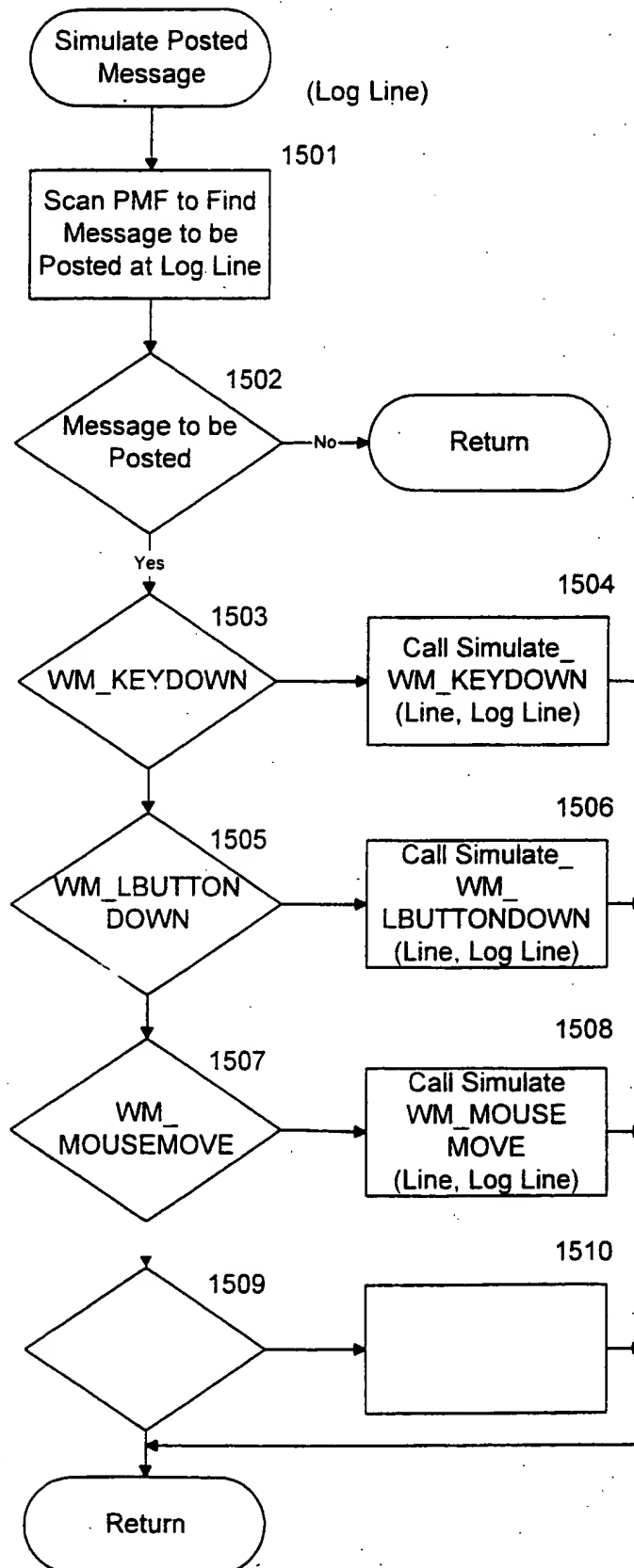


FIG. 13

**FIG. 14**

**FIG. 15**

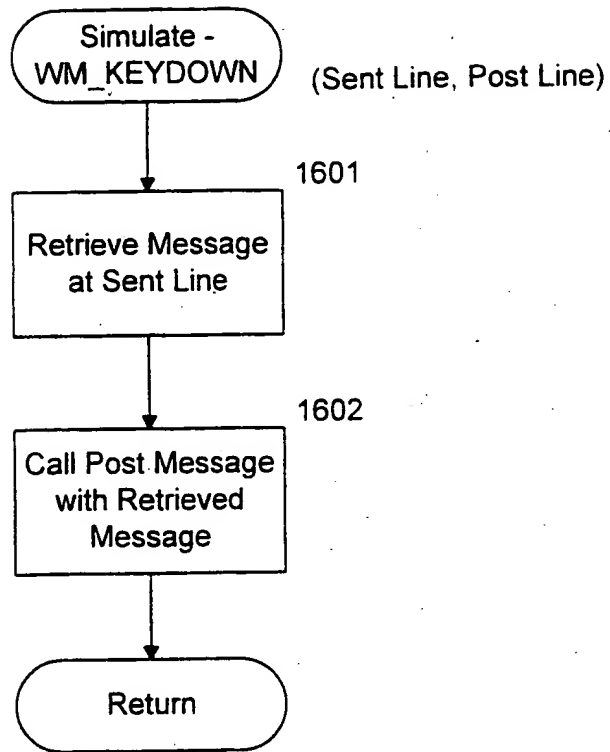
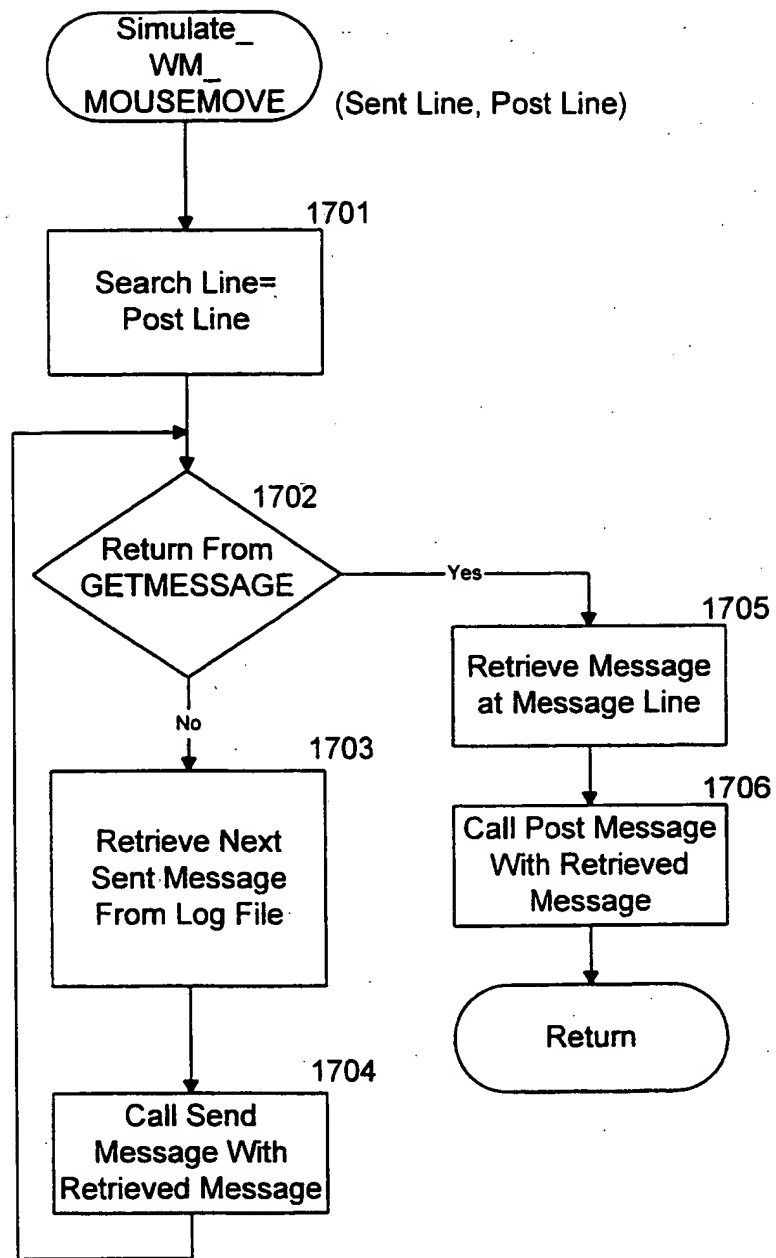
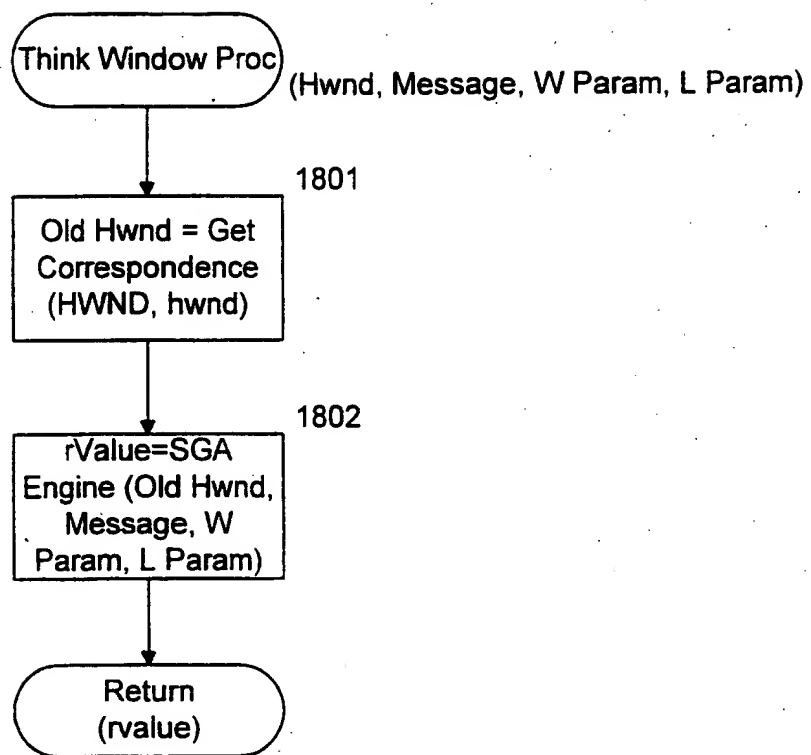


FIG. 16

**FIG. 17**

**FIG. 18**

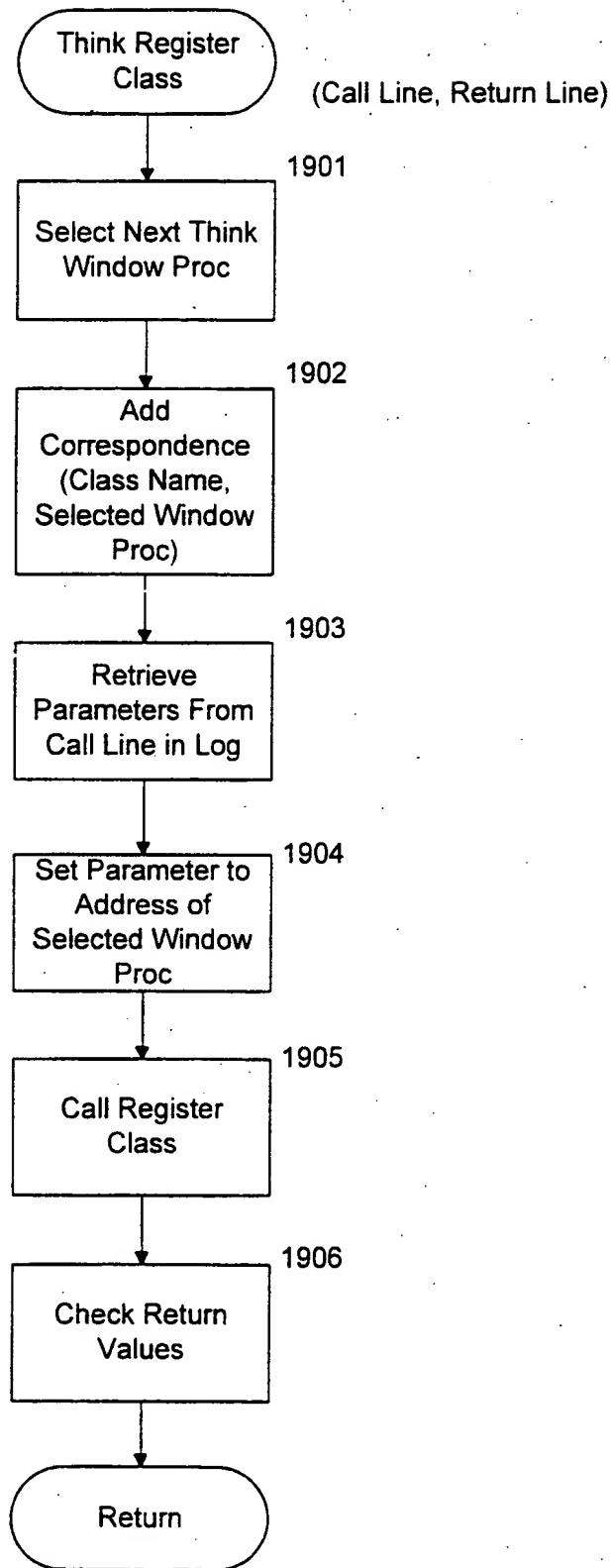
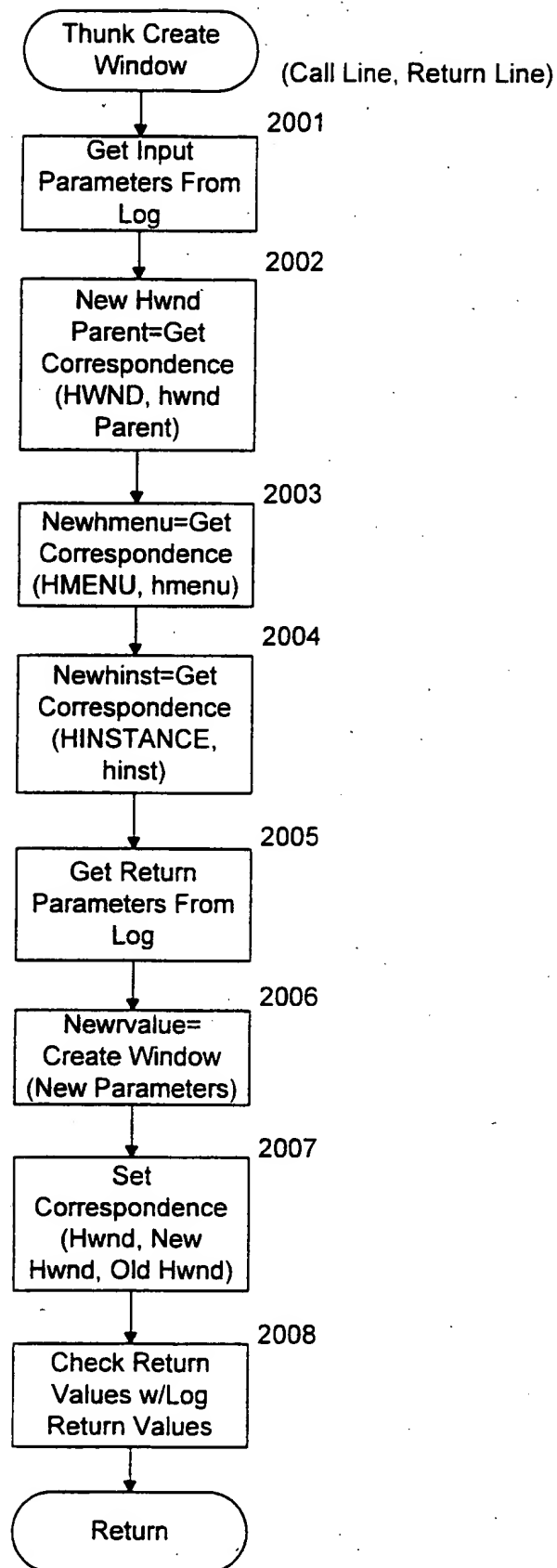


FIG. 19

**FIG. 20**

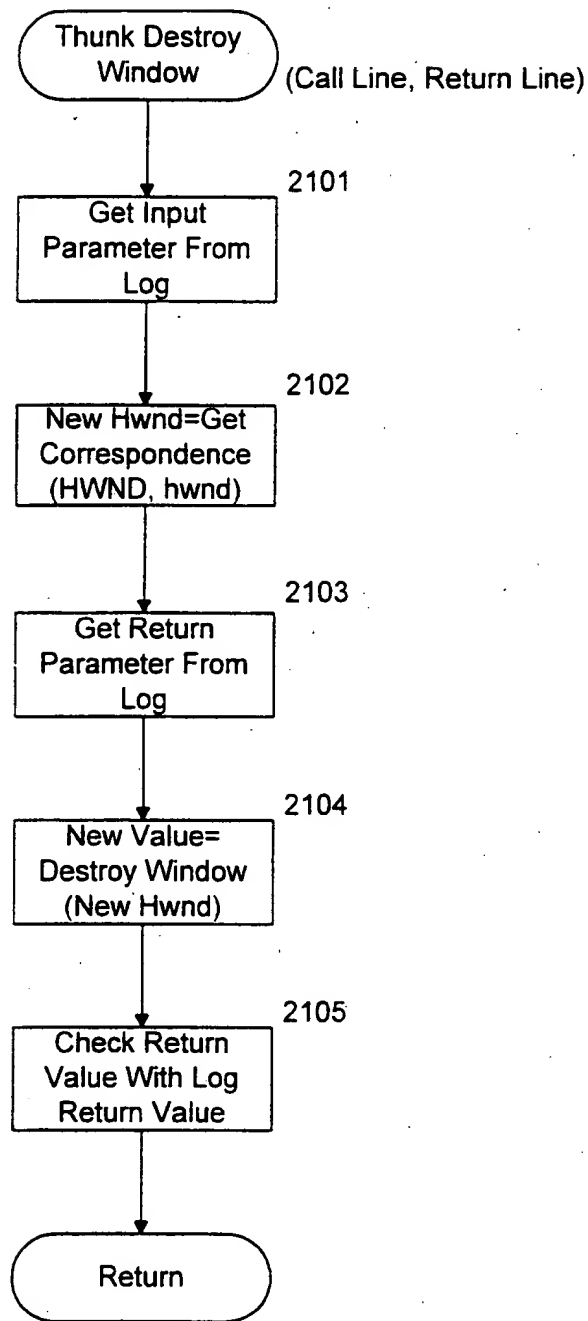


FIG. 21

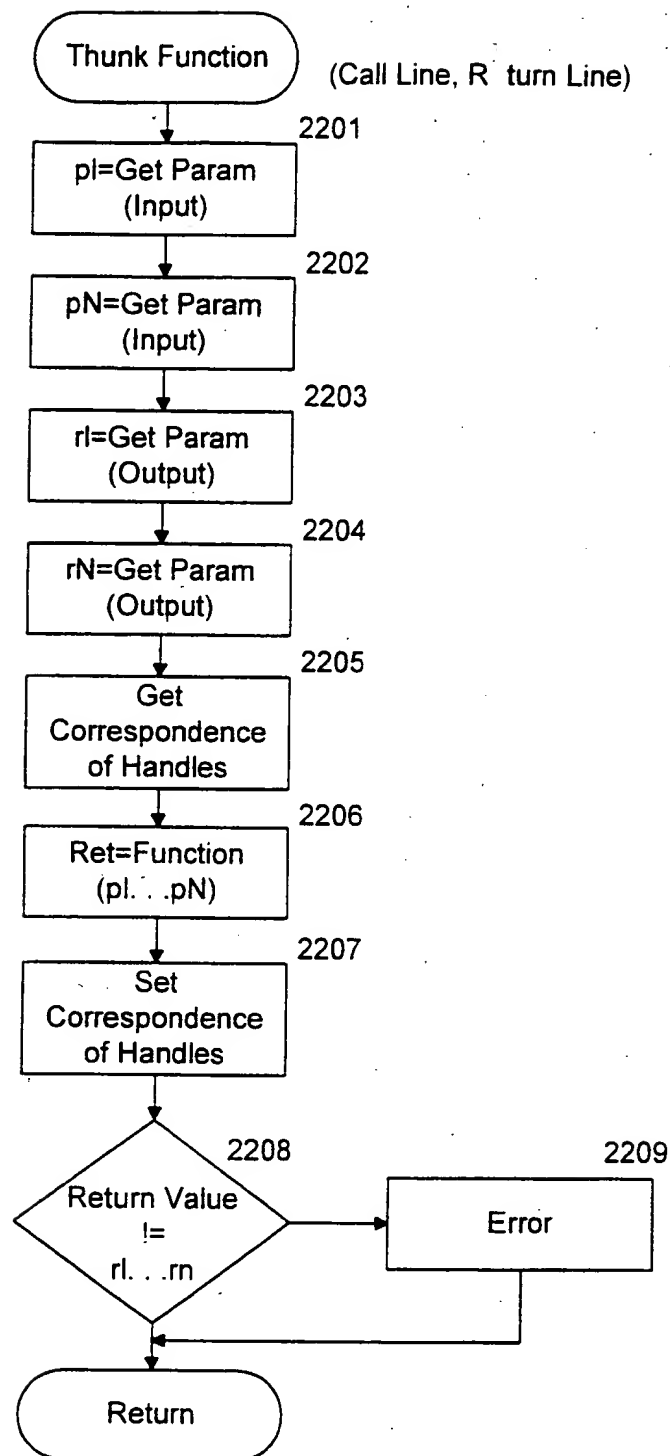


FIG. 22

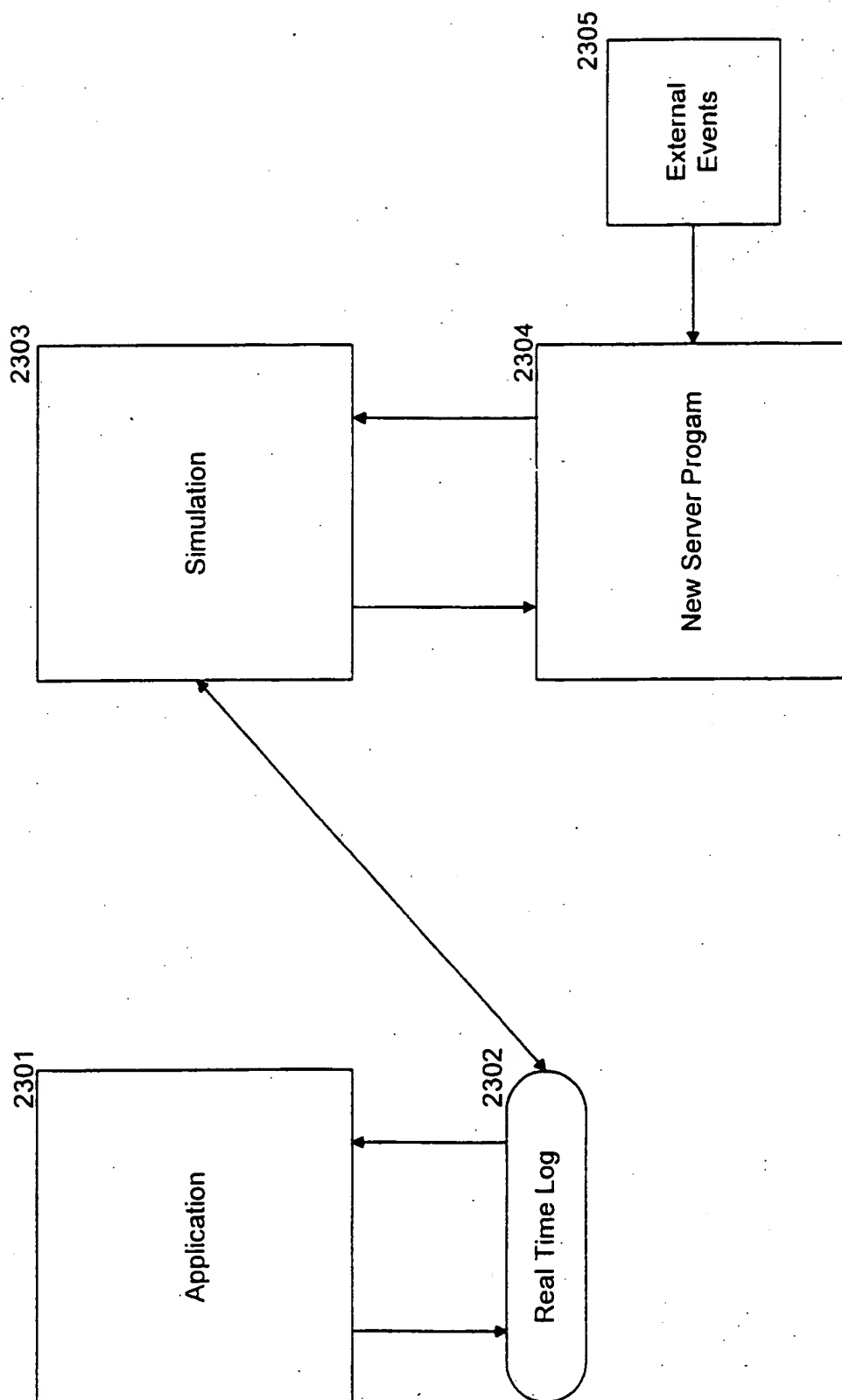


FIG. 23



European Patent
Office

EUROPEAN SEARCH REPORT

Application Number
EP 94 11 0016

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	DE-A-41 18 454 (SUN MICROSYSTEMS, INC.) * abstract *	1-26	G06F11/00
A	PROCEEDINGS OF THE 5TH ANNUAL EUROPEAN COMPUTER CONFERENCE, 16 May 1991, BOLOGNA pages 557 - 561 STEPHEN W.L. YIP ET AL. 'Applying formal specification and functional testing to graphical user interfaces' * page 560, right column, line 1 - line 16 *	1-26	
A	IBM TECHNICAL DISCLOSURE BULLETIN., vol.33, no.6B, November 1990, NEW YORK US pages 140 - 141 'Tracing the exported entry points in an OS/2 dynamic link library'	1,15,22	
			TECHNICAL FIELDS SEARCHED (Int.Cl.6)
			G06F
The present search report has been drawn up for all claims			
Place of search THE HAGUE		Date of completion of the search 28 September 1994	Examiner Corremans, G
CATEGORY OF CITED DOCUMENTS X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document			